НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.582

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Метод машинного навчання для аналізу результатів програмного забезпечення»

Виконав:
студент ІІ курсу, групи КП-81мп
Лисенко Олександр Олегович _____

Керівник: доцент кафедри ПЗКС, к.т.н.,
Доцент Олещенко Любов Михайлівна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри АУТС ФІОТ, к.т.н.,
доцент Полторак Вадим Петрович. _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення» («Програмне забезпечення комп'ютерних та інформаційно-пошукових систем»)

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____ І.А. Дичка

«___»_____ 2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Лисенку Олександру Олеговичу

1. Тема дисертації «Метод машинного навчання для аналізу результатів тестування програмного забезпечення», науковий керівник дисертації доцент кафедри ПЗКС, к.т.н., доцент Олещенко Любов Михайлівна затверджені наказом по університету від «13» листопада 2019 р. № 3895-с

2. Термін подання студентом дисертації «18» грудня 2019 р.

3. Об'єкт дослідження: процес автоматизації тестування програмного забезпечення за допомогою машинного навчання.

4. Предмет дослідження: є алгоритми кластеризації для аналізу результатів тестування програмного забезпечення.

5. Перелік завдань, які потрібно розробити:
   - оцінити сучасний стан проблеми, обґрунтувати актуальність напрямку досліджень, сформулювати мету та задачі дослідження;
   - проаналізувати методи кластеризації тексту за тематикою;
   - сформувати вимоги до власного методу для автоматичного аналізу результатів тестування;
   - описати принципи, що використовуються у методі: описати їх особливості та детально описати архітектури розроблюваної системи;
   - виконати тестування розробленого програмного забезпечення з використанням машинного навчання для автоматичного аналізу результатів тестування програмного забезпечення та проаналізувати отримані результати;
   - описати бізнес-модель, що дозволить представити на ринку повноцінний програмний продукт із використанням представлених у роботі напрацювань.
6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
   - схема алгоритму поєднання запусків;
   - дерево проблем;

7. Орієнтовний перелік публікацій:

- XII наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2019);

7. Дата видачі завдання «15» грудня 2018 р.

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., к.т.н., доцент		

9. Дата видачі завдання «25» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.11.2018	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	04.12.2018	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.02.2019	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	05.04.2019	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	15.05.2019	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	15.06.2019	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2019	13.11.2019	
8.	Оформлення текстової і графічної частини магістерської дисертації	05.12.2019	

Студент О.О Лисенко

Науковий керівник дисертації Л.М. Олещенко

РЕФЕРАТ

Актуальність теми. На сьогодні тестування займає один з ключових етапів розроблення програмного забезпечення (ПЗ). Для аналізу розроблюваного програмного забезпечення потрібні першокласні тестувальники, їх постійне навчання та мотивація, що для невеликих і середніх компаній є значною проблемою. Розробники програмного забезпечення намагаються уникати ручного тестування продукту з метою зниження ризику впливу людського фактору та зниження вартості тестування. Середній час аналізу помилок після автоматичного тестування для 500 «провалених» тестів може досягати до 24 год., тобто 4 робочі дні.

При розробленні продукту необхідно, щоб після його випуску він працював коректно, без збоїв та відповідав вимогам замовника. Бажано, щоб уже у процесі розроблення частково функціонуючий продукт працював згідно усіх вимог щодо функціональності та був масштабованим. Це завдання частоково допомагає вирішити тестування. IT-компанії потребують автоматизації процесу тестування, подальшого аналізу та виправлення помилок у ПЗ для надання замовнику повної картини того, что відбувається у проекті з детальним описом того, що вже працює та потребує доопрацювання. Це дуже складно робити вручну, особливо тоді, коли проект дуже великий і у ньому виконується більше тисячі тестів. В цьому допомогає автоматизація тестування, тобто написання автоматичних тестів, які можна буде виконувати повторно. Таким чином, після додавання нової функціональності в додаток можна бути впевненим, що все працює без збоїв, хоча і тут не все відбувається так гладко, як хотілося б.

Саме тому виникає потреба у розробленні програмної системи для автоматизованого аналізу результатів тестування програмного забезпечення, яка дозволить скоротити час аналізу помилок та переглядати стан виконання тестів у реальному часі.

Об’єктом дослідження є процес аналізу результатів тестування програмного забезпечення.

Предметом дослідження є алгоритми кластеризації для аналізу результатів тестування програмного забезпечення.

Мета роботи: створити програмний метод для аналізу результатів тестування програмного забезпечення на основі kNN-алгоритму.

Методи дослідження. В даній роботі використовуються ітеративні методи кластеризації, а саме: опорних векторів, метод Байєса та метод *k*-найближчих сусідів.

Наукова новизна роботи полягає в наступному:

1. Вперше застосований kNN-алгоритм для задачі аналізу результатів тестування програмного забезпечення, що дозволяє відносити текст результату тесту до категорій, заданих користувачем.

2. Вперше розроблено веб-додаток з використанням kNN-алгоритму для автоматичного аналізу результатів тестування.

3. **Практична цінність** отриманих в роботі результатів полягає в тому, що розроблену систему можна легко інтегрувати у існуюче програмне забезпечення завдяки використанню Docker-контейнера. Дана програмна система дозволяє зменшити час та кількість людських ресурсів, необхідних на аналіз помилок після проведення тестування та дозволяє менеджерам і замовникам слідкувати за станом проекту у реальному часі.

4. **Апробація роботи.** Основні положення і результати роботи були представлені та обговорювались на XII науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2019 (Київ, 13-15 листопада 2019 р.).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п'яти розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи.

У першому розділі було проаналізовано існуючі методи кластеризації для дослідження результатів тестування програмного забезпечення. Виявлено їх переваги та недоліки. На основі досліджень та результатів аналізу було сформовано вимоги до програмного забезпечення.

Основними вимогами стали: точність кластеризації; можливість створення категорій та підкатегорій для кластеризації помилок; можливість переглядати стан автоматизації тестування у режимі реального часу.

У другому розділі було детально описано роботу алгоритму відповідно до поставленої задачі. Основна ідея підходу полягає у тому, що використовується траса стеку у якості даних, які необхідно кластеризувати, адже саме ця частина має у собі інформацію про причину «падіння» тесту.

У третьому розділі описані вимоги користувача, функціональні вимоги та можливості запропонованого програмного забезпечення; наведено архітектуру, на основі якої виконувалось розроблення програмного забезпечення, інтерфейс користувача; проаналізовано сучасні мови програмування, середовища розробки та бібліотеки. Обрано стек технологій для реалізації методу.

У четвертому розділі описані результати порівняння методів кластеризації *k*-найближчих сусідів, опорних векторів та методу Байєса для автоматичного аналізу результатів тестування програмного забезпечення для 100, 200, 500 та 1000 «провалених» тестів. Проведено порівняння автоматичного аналізу результатів з ручним.

У п’ятому розділі створено бізнес-модель кінцевого продукту, виділено основні зацікавлені сторони у вирішенні існуючих недоліків, ступінь впливу даних сторін на вирішення проблем. Запропоновано комерційне рішення з конкурентними перевагами, що задовольняє інтереси зацікавлених осіб та виділено ціннісну пропозицію запропонованого продукту.

У висновках проаналізовано отримані результати роботи.

У додатках наведено архітектуру програмного забезпечення, а також дерево проблем.

Робота виконана на 89 аркушах, містить 2 додатки та посилання на список використаних літературних джерел з 37 найменувань. У роботі наведено 41 рисунок та 7 таблиць.

Ключові слова: тестування програмного забезпечення, машинне навчання, тест-кейс, stack trace, кластеризація, метод *k*-найближчих сусідів, Elasticsearch, TF, IDF.

ABSTRACT

Actuality of theme. Today, testing is one of the key stages of software development. Analyzing developed software requires first-class testers, their constant training and motivation, which is a big problem for small and medium-sized companies. Developers of software maintenance are trying to avoid manual testing product in order to reduce the risk exposure of the human factor and reduce the cost of testing . Average error analysis time after automatic Testing for 500 "failed" tests can reach up to 24 hours, i.e. 4 working days.

Is is very important, that after product release he should work correctly, without failures and meet the requirements of the customer . Preferably, that already in the process of drafting partially functioning product worked by all requirements regarding functionality and was scalable. This task often helps solve the test. IT-companies need to automate the process of testing, further analysis and correction of errors in the software providing the customer a complete picture of something happening in the project with detailed descriptions of what is already working and needs revision. It is very difficult to do manually, especially when when the project is very large and it performed more than a thousand tests. Automated testing can help to solve this problem. Write automated tests that can be carried out again so after adding new functionality to the application you can be sure that everything runs smoothly, but sometimes it is not going as smoothly as hoped.

That is why there is a need for development of software systems for automated analysis of results of testing software maintenance, which will reduce the time of analysis errors and view the status of execution of tests in a real time.

The object of the study is the process of automating software testing using machine learning methods.

The subject of the study is clustering algorithms for analyzing software test results.

Purpose: to create a software method for analyzing the results of software testing based on the kNN algorithm.

Research methods. This paper uses methods of theoretical research: analysis, synthesis and generalization. Empirical methods were also used: experiment, observation, measurement and description.

The scientific novelty of the work is as follows:

1. A software method for automatic analysis of software test results using kNN algorithm is developed , which allows to relate the text of the test result to user-defined categories .
2. A web application was developed using the method for automatic analysis of test results, which will significantly reduce the time and amount of human resources, required to analyze errors after testing, and allow managers and clients to monitor the state of project automation in real time .

The practical value of the results obtained is that the developed system can be easily integrated into existing software thanks to the Docker container. The system will also significantly reduce the time and amount of human resources, required to analyze errors after testing, and allow managers and clients to monitor the state of the project automation in real time.

Approbation. The main provisions and results of the work were presented and discussed at the XII Scientific Conference of Undergraduate and Graduate Students in Applied Mathematics and Computing PMK-2019 (Kyiv, November 13-15, 2019).

Structure and content of the thesis. The master's thesis consists of an introduction, five sections, conclusions and appendices.

The introduction gives a general description of the work, assesses the current state of the problem, substantiates the relevance of the research direction, formulates the purpose and objectives of the research, shows the scientific novelty of the results obtained and the practical value of the work.

The first section analyzes existing clustering methods to investigate software test results. Their advantages and disadvantages are revealed. Based on the research and analysis results, software requirements have been generated.

The main requirements were: clustering accuracy; ability to create categories and subcategories for clustering errors; the ability to view the state of test automation in real time.

The second section was detailed operation algorithm according to the task. The basic idea behind the approach is that the stack path is used as data to be clustered , because this part contains information about the cause of the test crash.

The third section describes the user requirements, functional requirements and capabilities of the developed system; shows the architecture based on the software development, user interface; modern programming languages, development environments, and libraries were analyzed. Technology stack for method implementation was selected.

The fourth section describes the results of comparing the clustering methods of k-nearest neighbors, reference vectors, and Bayes to automatically analyze software test results for 100, 200, 500, and 1000 failed tests. Automatic comparison of results with manual was also compared.

The fifth section was analyzed the current situation in the field of automated testing revealed problems and summarizes them in an appropriate tree problems. Along with the problems, the business model of the final product was formed and built, the main stakeholders were identified in solving the existing shortcomings, the degree of influence of these parties on the problem solving.

The conclusion is analyzed the results of this master's work.

The applications provide a architecture of the developed system as well as a problem tree.

The work is made on 89 sheets, contains 2 applications and links to the list of used literature sources of 37 titles. The work contains 41 figures and 7 tables.

Keywords: software testing, machine learning, test case, stack trace, clustering, *k*-means method, Elasticsearch, TF, IDF.

РЕФЕРАТ

Актуальность темы. Сегодня тестирование занимает один из ключевых этапов разработки программного обеспечения. Для анализа разрабатываемого программного обеспечения нужны первоклассные тестировщики, их постоянное обучение и мотивация, для небольших и средних компаний является значительной проблемой. Разработчики программного обеспечения стараются избегать ручного тестирования продукта с целью снижения риска воздействия человеческого фактора и снижение стоимости тестирования. Средний время анализа ошибок после автоматического тестирование для 500 «провалившихся» тестов может достигать до 24 ч., то есть 4 рабочих дня.

При разработке продукта необходимо, чтобы после его выпуска он работал корректно, без сбоев и отвечал требованиям заказчика. Желательно, чтобы уже в процессе разработки частично функционирующий продукт работал согласно всех требований по функциональности и был масштабируемым. Эту задачу частично помогает решить тестирования. IT-компании требуют автоматизации процесса тестирования, дальнейшего анализа и исправления ошибок в ПО для предоставления заказчику полной картины того, что происходит в проекте с подробным описанием того, что уже работает и требует доработки. Это очень сложно делать вручную, особенно тогда, когда проект очень большой и в нем выполняется более тысячи тестов. В этом помогает автоматизация тестирования, то есть написание автоматических тестов, которые можно будет выполнять заново, таким образом, после добавления новой функциональности в приложение, можно быть уверенным, что все работает без сбоев, хотя и здесь не все происходит так гладко как хотелось.

Объектом исследования является процесс автоматизации тестирования программного обеспечения с помощью методов машинного обучения.

Предметом исследования является алгоритмы кластеризации для анализа результатов тестирования программного обеспечения.

Цель работы: создать программный метод для автоматического анализа результатов тестирования программного обеспечения на основе алгоритма kNN.

Методы исследования. В данной работе используются методы теоретического исследования: анализ, синтез и обобщение. Также применялись эмпирические методы: эксперимент, наблюдение, измерение и описание.

Научная новизна работы заключается в следующем:

1. Разработан программный метод для автоматического анализа результатов тестирования программного обеспечения с использованием алгоритма kNN, что позволяет относить текст результата теста к категориям, заданных пользователем.
2. Разработан веб-приложение с использованием метода для автоматического анализа результатов тестирования, что значительно уменьшит время и количество человеческих ресурсов, необходимых на анализ ошибок после тестирования, и позволит менеджерам и заказчикам следить за состоянием автоматизации проекта в режиме реального времени.

Практическая ценность полученных в работе результатов заключается в том, что разработанную систему можно легко интегрировать в существующее программное обеспечение благодаря Docker-контейнеру. Также данная система значительно уменьшит время и количество человеческих ресурсов, необходимых на анализ ошибок после тестирования, и позволит менеджерам и заказчикам следить за состоянием автоматизации проекта в режиме реального времени.

Апробация работы. Основные положения и результаты работы были представлены и обсуждались на ХИИ научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» ПМК-2019 (Киев, 13-15 ноября 2019).

Структура и объем работы. Магистерская диссертация состоит из введения, пяти глав, заключения и приложений.

Во введении дана общая характеристика работы, выполнена оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулированы цели и задачи исследований, показано научную новизну полученных результатов и практическую ценность работы.

В первом разделе были проанализированы существующие методы кластеризации для исследования результатов тестирования программного обеспечения. Выявлены их преимущества и недостатки. На основе исследований и результатов анализа был сформирован требования к программному обеспечению.

Основными требованиями стали: точность кластеризации; возможность создания категорий и подкатегорий для кластеризации ошибок; возможность просматривать состояние автоматизации тестирования в режиме реального времени.

Во втором разделе было подробно описано работу алгоритма в соответствии с поставленной задачи. Основная идея подхода состоит в том, что используется трасса стека в качестве данных, которые необходимо кластеризовать, ведь именно эта часть имеет в себе информацию о причине падения теста.

В третьем разделе описаны требования пользователя, функциональные требования и возможности разработанной системы; приведена архитектура, в основе которой выполнялась разработка программного обеспечения, интерфейс пользователя; проанализированы современные языки программирования, среды разработки и библиотеки. Выбран стек технологий для реализации метода.

В четвертом разделе были описаны и результаты сравнения методов кластеризации k-ближайших соседей, опорных векторов и метода Байеса для автоматического анализа результатов тестирования программного

обеспечения на 100, 200, 500 и 1000 «провалившихся» тестов. Также было проведено сравнение автоматического анализа результатов с ручным.

В пятом разделе было проведен анализ текущей ситуации в сфере автоматизации тестирования, выявлено имеющиеся проблемы и подведены их в соответствующем дереве проблем. Наряду с проблемами было сформировано и построено бизнес-модель конечного продукта, выделены основные заинтересованные стороны в решении существующих недостатков, степень влияния данных сторон на решение проблем.

В выводах проанализированы полученные результаты работы.

В приложениях приведены архитектура разработанной системы, а также дерево проблем.

Работа выполнена на 89 листах, содержит 2 вложения и ссылки на список использованих литературных источников из 37 наименований. В работе приведены 41 рисунок и 7 таблиц.

Ключевые слова: тестирование программного обеспечения, машинное обучение, тест-кейс, трассировка стека, кластеризация, метод k-ближайших соседей, Elasticsearch, TF, IDF.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Stack trace – список методів, які були викликані до моменту, коли в додатку відбулася помилка.

Кластеризація – задача розбиття заданої вибірки об'єктів (ситуацій) на підмножини, які називаються кластерами, так, щоб кожен кластер складався зі схожих об'єктів, а об'єкти різних кластерів істотно відрізнялися.

TF (term frequency – частота слова) – відношення числа входжень обраного слова до загальної кількості слів документа. Оцінюється важливість слова у межах обраного документа.

IDF (inverse document frequency – обернена частота документа) – інверсія частоти, з якою слово зустрічається в документах колекції.

Фреймворк – це реальна або концептуальна структура, призначена слугувати опорою або керівництвом для побудови чогось, що розширює структуру на щось корисне.

API (Application Programming Interface) – це набір публічних методів та властивостей, які використовуються для взаємодії з іншими об’єктами у програмі.

Angular – JavaScript фреймворк для створення клієнтської частини веб-додатку.

HTML (Hyper Text Markup Language) – мова гіпертекстової розмітки.
DOM (Document Object Model) – програмний інтерфейс, що не залежить від платформи та мови, який дозволяє програмам та скриптам отримувати доступ до вмісту HTML-подібних документів та змінювати його.

ВСТУП

На сьогодні тестування займає один з ключових етапів розроблення програмного забезпечення. Для аналізу розроблюваного програмного забезпечення потрібні першокласні тестувальники, їх постійне навчання та мотивація, що для невеликих і середніх компаній є значною проблемою. Розробники програмного забезпечення намагаються уникати ручного тестування продукту з метою зниження ризику впливу людського фактору та зниження вартості тестування. Середній час аналізу помилок після автоматичного тестування для 500 "провалених" тестів може досягати до 24 год, тобто 4 робочі дні.

При розробленні продукту необхідно, щоб після його випуску він працював коректно, без збоїв та відповідав вимогам замовника. Бажано, щоб уже у процесі розроблення частково функціонуючий продукт працював згідно усіх вимог щодо функціональності та був масштабованим. Це завдання частоково допомагає вирішити тестування. ІТ-компанії потребують автоматизації процесу тестування, подальшого аналізу та виправлення помилок у ПЗ для надання замовнику повної картини того, що відбувається у проекті з детальним описом того, що вже працює та потребує доопрацювання. Це дуже складно робити вручну, особливо тоді, коли проект дуже великий і у ньому виконується більше тисячі тестів. В цьому допомогає автоматизація тестування, тобто написання автоматичних тестів, які можна буде виконувати заново. Таким чином, після додавання нової функціональності в додаток можна бути впевненим, що все працює без збоїв, хоча і тут не все відбувається так гладко як хотілося.

Виникають проблеми з менеджерської сторони відносно аналізу результатів тестування програмного зебезпечення. Покриття тестами зростає надто повільно і ні замовник, ні команда цього не бачить, а потім через півроку з’ясовується, що замовник сподівався на вдвічі більше покриття. Іноді відбувається так, що тести були реалізовані, але не запущені, гроші витрачено на цю роботу, але цінність нульова. Наступний момент,

коли тести були написані, запущені, але не проаналізовані, також цінність нульова, так як гроші було витрачено, але результат ніхто не перевірив. Також написані тести можуть бути нестабільними і команда витрачає час на виправлення цих тестів та повторний їх запуск, витрачається в два рази більше грошей, що також не дуже добре для замовника та бізнесу.

Тому виникає потреба у розробленні програмної системи для автоматизованого аналізу результатів тестування програмного забезпечення, яка дозволить скоротити час аналізу помилок та переглядати стан виконання тестів у режимі реального часу.

1. АНАЛІЗ МЕТОДІВ КЛАСТЕРИЗАЦІЇ

1.1. Аналіз існуючих методів кластеризації

Кластеризація (або кластерний аналіз) – це задача розбиття множини об'єктів на групи, які називаються кластерами. Усередині кожної групи повинні виявитися "схожі" об'єкти, а об'єкти різних груп повинні бути якомога більш відмінними. Головна відмінність кластеризації від класифікації полягає в тому, що перелік груп чітко не заданий і визначається в процесі роботи алгоритму.

Кластер має наступні математичні характеристики: центр, радіус, середнє квадратичне відхилення, розмір кластера.

Центр кластера – це середнє геометричне місце точок у просторі змінних. Радіус кластера – максимальна відстань точок від центру кластера.

Кластери можуть перекриватися. У цьому випадку неможливо за допомогою математичних процедур однозначно віднести об'єкт до одного з двох кластерів. Такі об'єкти називають спірними.

Спірний об'єкт – це об'єкт, який у міру подібності може бути віднесений до кількох кластерів.

Розмір кластера може бути визначений або по радіусу кластера, або по середньоквадратичному відхиленню об'єктів цього кластера. Об'єкт відноситься до кластеру, якщо відстань від об'єкта до центру кластера менше радіуса кластера. Якщо ця умова виконується для двох і більше кластерів, об'єкт є спірним. Неоднозначність даного завдання може бути усунена експертом або аналітиком.

Робота кластерного аналізу спирається на два припущення. Перше припущення – розглядувані ознаки об'єкта допускають бажане розбиття пулу (сукупності) об'єктів на кластери. Друге припущення – правильність вибору масштабу або одиниць вимірювання ознак.

Застосування кластерного аналізу в загальному вигляді зводиться до:

– відбір вибірки об'єктів для кластеризації;

– визначення множини змінних, за якими будуть оцінюватися об'єкти у вибірці. При необхідності – нормалізація значень змінних;
– обчислення значень міри схожості між об'єктами. Застосування методу кластерного аналізу для створення груп схожих об'єктів (кластерів);
– представлення результатів аналізу.

Після отримання та аналізу результатів можливе коригування обраної метрики і методу кластеризації до отримання оптимального результату.

Отже, як же визначати "схожість" об'єктів? Для початку потрібно скласти вектор характеристик для кожного об'єкта – як правило, це набір числових значень, наприклад, зріст, вага людини тощо. Однак існують також алгоритми, що працюють з якісними (т.зв. категорійними) характеристиками. Після того, як ми визначили вектор характеристик, можна провести нормалізацію, щоб всі компоненти давали однаковий внесок при розрахунку "відстані". У процесі нормалізації все значення приводяться до деякого діапазону, наприклад, $[1, -1]$ або $[0, -1]$. Нарешті, для кожної пари об'єктів вимірюється "відстань" між ними – ступінь схожості. Існує багато метрик, розглянемо основні з них.

Евклідова відстань. Найбільш поширена функція відстані. Являє собою геометричну відстань у багатовимірному просторі [4]:

$$\rho(x, x') = \sqrt{\sum_i^n (x_i - x_i')^2}. \quad (1)$$

Квадрат евклідової відстані. Застосовується для додання більшої ваги більш віддаленим один від одного об'єктів. Це відстань обчислюється таким чином [4]:

$$\rho(x, x') = \sum_i^n (x_i - x_i')^2. \quad (2)$$

Манхетенська відстань [4]. Це відстань є усередненим значенням різниць по координатах [4]:

$$\rho(x, x') = \sum_i^n |x_i - x_i'|^2. \quad (3)$$

У більшості випадків ця міра відстані приводить до таких же результатів, як і для звичайної відстані Евкліда. Однак для цієї міри вплив окремих великих різниць (викидів) зменшується (тому що вони не зводяться в квадрат).

Відстань Чебишева [5]. Ця відстань може виявитися корисною, коли потрібно визначити два об'єкти як "різні", якщо вони різняться якоюсь однією координатою.

Степенева відстань [5]:

$$\rho(x, x') = \sum_i^n |x_i - x_i'|. \quad (4)$$

Застосовується в разі, коли необхідно збільшити або зменшити вагу, що відноситься до розмірності, для якої відповідні об'єкти сильно відрізняються. Степенева відстань [6] обчислюється за такою формулою [5]:

$$\rho(x, x') = \sqrt[r]{\sum_i^n |x_i - x_i'|^p}, \quad (5)$$

де r і p – параметри, що визначаються користувачем. Параметр p відповідає за поступове зважування різниць за окремими координатами, параметр r відповідає за прогресивне зважування великих відстаней між об'єктами. Якщо обидва параметри – r і p – дорівнюють двом, то це відстань збігається з відстанню Евкліда.

Характерно, що вибір метрики повністю залежить від потреб дослідника, оскільки результати кластеризації можуть істотно відрізнятися при використанні різних методів.

У наступних пунктах ми розглянемо найпоширеніші методи кластеризації даних з розрахунку на те, щоб вони працювали належним чином для великих об’ємів даних. Всі методи можна умовно поділити на 3 основні категорії, а саме:

- ітеративні;
- ієрархічні;
- щільнісні (Density-based).

1.2. Аналіз методів кластеризації для дослідження результатів тестування програмного забезпечення

1.2.1. *Ітеративні методи*

Ітеративні методи виявляють більш високу стійкість по відношенню до шумів і викидів, некоректного вибору метрики, включенню незначущих змінних в набір, який бере участь в кластеризації. Ціною, яку доводиться "платити" за ці переваги методу, є слово "апріорі". Аналітик повинен заздалегідь визначити кількість кластерів, кількість ітерацій або правило зупинки, а також деякі інші параметри кластеризації. Це особливо складно початківцям фахівцям.

Якщо немає припущень щодо числа кластерів, доцільно використовувати ієрархічні алгоритми. Однак, якщо обсяг вибірки не дозволяє це зробити, можливий шлях – проведення ряду експериментів з різною кількістю кластерів, наприклад, почати розбиття сукупності даних з двох груп і, поступово збільшуючи їх кількість, порівнювати результати. За рахунок такого "варіювання" результатів досягається досить велика гнучкість кластеризації.

Алгоритм *k*-середніх (*k*-means).

Найбільш поширений серед ітеративних методів алгоритм *k*-середніх, також званий швидким кластерним аналізом. На відміну від ієрархічних методів, які не вимагають попередніх припущень щодо числа кластерів, для можливості використання цього методу необхідно мати гіпотезу про найбільш ймовірну кількість кластерів. Алгоритм *k*-середніх будує *k* кластерів, розташованих на великих відстанях один від одного. Основний тип задач, які вирішує алгоритм *k*-середніх, – наявність припущень (гіпотез) щодо числа кластерів, при цьому вони повинні бути різні настільки, наскільки це можливо. Вибір числа *k* може базуватися на результатах попередніх досліджень, теоретичних міркуваннях або інтуїції.

Метод базується на мінімізації суми квадратів відстаней між кожним

спостереженням та центром його кластера, тобто функції. Використовується набір векторів, що належать до *i*-го кластеру, та середнє значення цих векторів.

Основна ідея полягає в тому, що на кожній ітерації заново вираховується центр мас для кожного кластера, потім вектори розбиваються на нові класи, відповідно до того, який з отриманих центрів виявився ближчим за метрикою (приклад алгоритму на рис. 1.1).

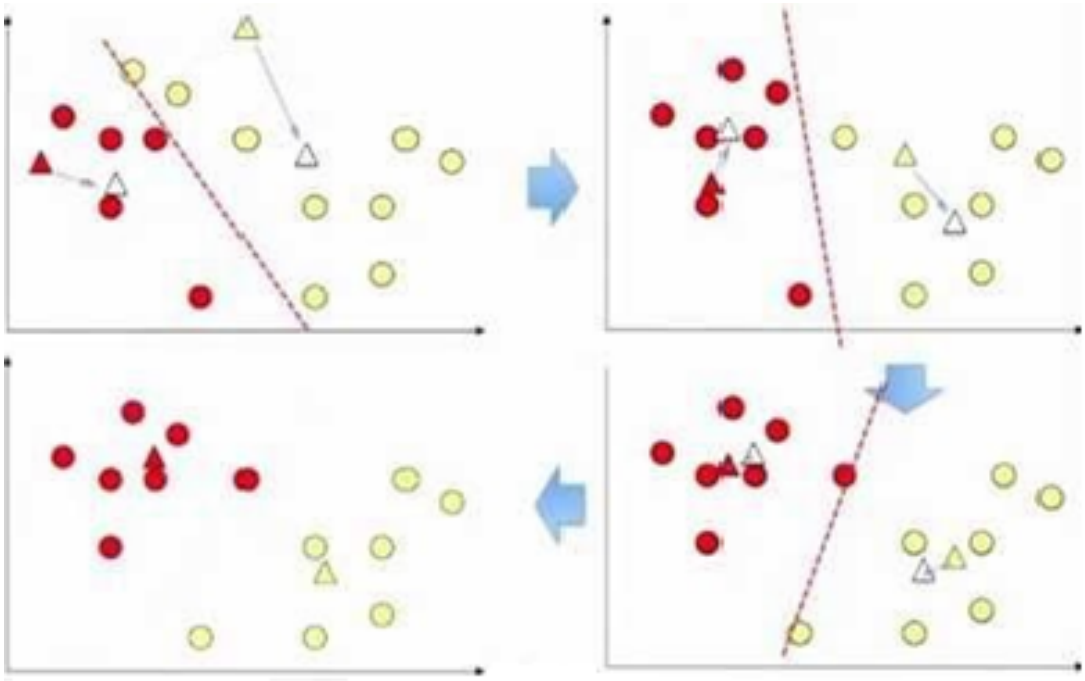


Рис. 1.1. Приклад *k*-means кластеризації [1]

Переваги алгоритму *k*-середніх:

- простота використання;
- швидкість використання;
- зрозумілість і прозорість алгоритму.

Недоліки алгоритму *k*-середніх:

- алгоритм занадто чутливий до викидів, які можуть спотворювати середнє. Можливим вирішенням цієї проблеми є використання модифікації алгоритму – алгоритм *k*-медіани;
- алгоритм може повільно працювати на великих наборах даних.

Можливим вирішенням цієї проблеми є використання вибірки даних.

Метод найближчого сусіда (kNN-алгоритм)

Метод найближчого сусіда являється одним із найпростіших алгоритмів кластеризації. Об’єкт x, що кластеризується, відноситься до класу y_i, якому належить найближчий об’єкт навчальної вибірки x_i. Для підвищення надійності класифікації об'єкт відноситься до того класу, якому належить більшість з його сусідів – k найближчих до нього об’єктів навчальної вибірки x_i. У задачах з двома класами число сусідів беруть непарним, щоб не виникало ситуацій неоднозначності, коли однакове число сусідів належать різним класам.

У здачах з числом класів 3 і більше непарність вже не допомагає, і ситуації неоднозначності все одно можуть виникати. Тому i-му сусіду приписуються ваги ω_i. Об'єкт відноситься до того класу, який набирає найбільші сумарні ваги серед k найближчих сусідів.

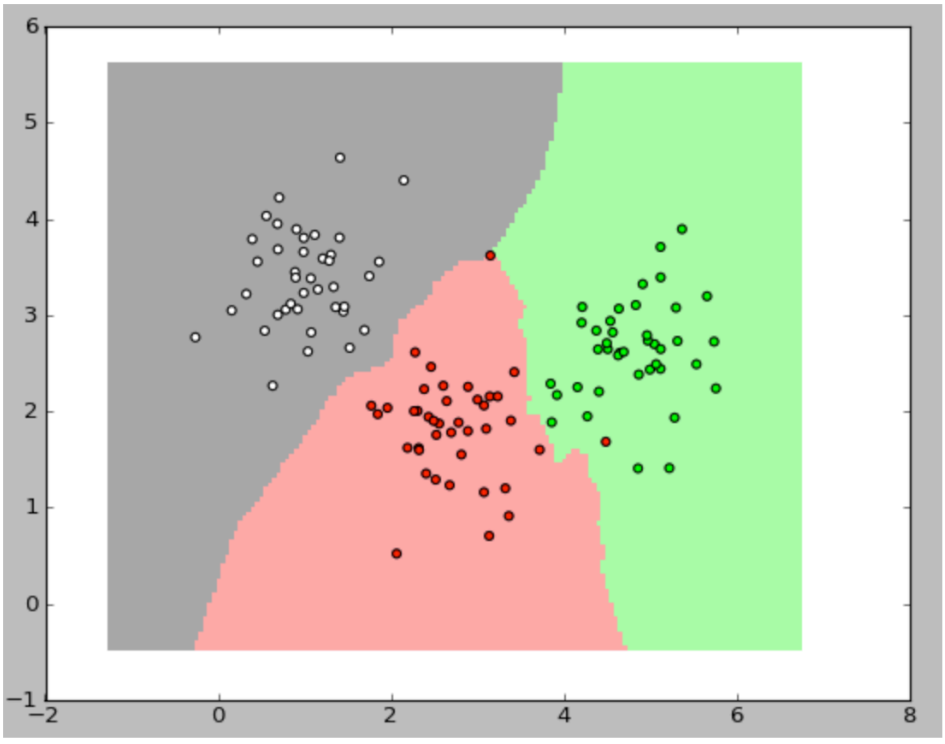


Рис. 1.2. Приклад навчальної вибірки kNN-алгоритму

Усі перераховані методи неявно спираються на одне важливе припущення, зване гіпотезою компактності: якщо міра подібності об'єктів введена досить вдало, то схожі об'єкти набагато частіше лежать в одному класі, ніж в різних. В цьому випадку межа між класами має досить просту форму, а класи утворюють компактно локалізовані області в просторі об'єктів. Зауважимо, що в математичному аналізі компактними називаються обмежені замкнуті множини.

Алгоритм EM (Expectation-maximization)

EM-алгоритм – алгоритм, що використовується для знаходження оцінок максимальної схожості параметрів ймовірних моделей, у випадку, коли модель залежить від деяких прихованих змінних. Кожна ітерація алгоритму складається з двох кроків. На E-кроці (expectation) вираховується очікуване значення функції правдоподібності, при цьому приховані змінні розглядаються як спостережувані. На M-кроці (maximization) вираховується оцінка максимальної схожості, таким чином, збільшується очікувана схожість, вирахувана на E-кроці. Потім це значення використовується для E-кроку на наступній ітерації. Алгоритм виконується до збіжності.

Алгоритм EM простий в реалізації, не чутливий до ізольованих об'єктів і швидко сходиться при вдалій ініціалізації. Однак він вимагає для ініціалізації вказівки кількості кластерів k, що має на увазі наявність апріорних знань про дані. Крім того, при невдалій ініціалізації збіжність алгоритму може виявитися повільною або може бути отриманий неякісний результат.

Алгоритм PAM (partitioning around Medoids)

PAM є модифікацією алгоритму k-середніх, алгоритмом k-медіани (k-medoids). Алгоритм менш чутливий до шумів і викидів даних, ніж алгоритм k-means, оскільки медіана менше піддається впливам викидів. PAM ефективний для невеликих наборів даних, його не бажано

використовувати для великих наборів даних.

Більш зрозумілі і прозорі результати кластеризації можуть бути отримані, якщо замість множини вихідних змінних використовувати якісь узагальнені змінні або критерії, які містять в стислому вигляді інформацію про зв'язки між змінними. Тобто виникає задача зниження розмірності даних. Вона може вирішуватися за допомогою різних методів, один з найбільш поширених – факторний аналіз.

Факторний аналіз переслідує дві мети:

– скорочення числа змінних;
– класифікацію змінних – визначення структури взаємозв'язків між змінними.

На першому кроці факторного аналізу здійснюється стандартизація значень змінних. Факторний аналіз спирається на гіпотезу про те, що аналізовані змінні є непрямими проявами порівняно невеликого числа прихованих чинників.

Факторний аналіз – це сукупність методів, орієнтованих на виявлення та аналіз прихованих залежностей між спостережуваними змінними. Приховані залежності також називають латентними.

Один з методів факторного аналізу – метод головних компонент – заснований на припущенні про незалежність факторів один від одного.

1.2.2. *Ієрархічні методи*

Загальна ідея методів даної групи полягає в послідовній ієрархічній декомпозиції множини об'єктів. У разі агломеративного методу декомпозиції (від низу до верху) кожен об'єкт являє собою самостійний кластер. На кожній ітерації пари прилеглих кластерів послідовно об'єднуються в загальний кластер. Ітерації тривають до тих пір, поки всі об'єкти не будуть об'єднані в один кластер або поки не виконається деяка умова зупинки. Низхідний метод (зверху вниз) навпаки, ґрунтується на

тому, що на початковому етапі всі об'єкти об'єднані в єдиний кластер. На кожній ітерації він розділяється на більш дрібні до тих пір, поки кожен об'єкт не виявиться в окремому кластері або не буде виконано умову зупинки. Як умова зупинки можна використовувати граничне число кластерів, яке необхідно отримати, проте зазвичай використовується порогове значення відстані між кластерами.

Основна проблема ієрархічних методів полягає в складності визначення умови зупинки таким чином, щоб виділити "природні" кластери і в той же час не допустити їх розбиття. Ще одна проблема ієрархічних методів кластеризації полягає у виборі точки поділу або злиття кластерів. Цей вибір критичний, оскільки після поділу або злиття кластерів на кожному наступному кроці метод буде оперувати тільки новоствореними кластерами, тому невірний вибір точки злиття або поділу на будь-якому етапі може привести до неякісної кластеризації. Крім того, ієрархічні методи не можуть бути застосовані до великих наборах даних, тому рішення про поділ або злиття кластерів вимагає аналізу великої кількості об'єктів і кластерів, що веде до великої обчислювальної складності методу.

Алгоритм BIRCH

Завдяки узагальненому вигляду кластерів, швидкість кластеризації збільшується, алгоритм BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) при цьому володіє великим масштабуванням. У цьому алгоритмі реалізований двоетапний процес кластеризації.

В ході першого етапу формується попередній набір кластерів. На другому етапі до виявлених кластерів застосовуються інші алгоритми кластеризації, придатні для роботи в оперативній пам'яті.

Розглянемо аналогію, що описує цей алгоритм. Якщо кожен елемент даних уявити як намистину, що лежить на поверхні столу, то кластери намистин можна "замінити" тенісними кульками і перейти до більш детального вивчення кластерів тенісних кульок. Число намистин може

виявитися досить велике, проте діаметр тенісних кульок можна підібрати таким чином, щоб на другому етапі можна було, застосувавши традиційні алгоритми кластеризації, визначити дійсну складну форму кластеру.

Алгоритм WaveCluster

WaveCluster є алгоритмом кластеризації, що ґрунтується на основі хвильових перетворень. На початку роботи алгоритму дані узагальнюються шляхом накладення на простір даних багатовимірної решітки. На подальших кроках алгоритму аналізуються не окремі точки, а узагальнені характеристики точок, які потрапили в одну клітинку решітки. В результаті такого узагальнення необхідна інформація вміщується в оперативній пам'яті. На наступних кроках для визначення кластерів алгоритм застосовує хвильове перетворення до узагальнених даних.

Головні особливості WaveCluster:

- складність реалізації;
- алгоритм може виявляти кластери довільних форм;
- алгоритм нечутливий до шумів;
- алгоритм може бути застосований тільки до даних низької розмірності.

Алгоритм CLARA

Алгоритм CLARA (Clustering LARge Applications) "витягує" множину зразків з бази даних. Кластеризація застосовується до кожного із зразків, на виході алгоритму пропонується найкраща кластеризація.

Для великих баз даних цей алгоритм ефективніший, ніж алгоритм PAM. Ефективність алгоритму залежить від обраного в якості зразка набору даних. Хороша кластеризація на обраному наборі може не дати хорошу кластеризацію на всій множині даних.

1.2.3. *Щільнісні методи*

Дана категорія методів розглядає кластери, як ділянки простору даних з високою щільністю об'єктів, які розділені ділянками з низькою щільністю об'єктів. Щільнісні методи часто застосовуються для фільтрації шуму та виявлення кластерів довільної форми.

Алгоритм DBSCAN

Алгоритм DBSCAN – один з перших алгоритмів кластеризації щільнісних методів. В основі цього алгоритму лежить кілька визначень:

1. ε-околицею об'єкта називається околиця радіуса ε деякого об'єкту.
2. Кореневим об'єктом називається об'єкт, ε-околиця якого містить не менше деякого мінімального числа MinPts об'єктів.
3. Об'єкт p безпосередньо щільно-досяжний з об'єктом q, якщо p знаходиться в ε-околиці q і q є кореневим об'єктом.
4. Об'єкт p щільно-досяжний з об'єкта q при заданих ε і MinPts, якщо існує послідовність об'єктів p_1, ..., p_n, де $p_1 = q$ і $p_1 = p$, така що p_{1+n} безпосередньо щільно досяжний з p_i, $1 \leq i \leq n$.
5. Об'єкт p щільно-з'єднаний з об'єктом q при заданих ε і MinPts, якщо існує об'єкт o такий, що p і q щільно-досяжні з o.

Для пошуку кластерів алгоритм DBSCAN перевіряє ε-околицю кожного об'єкта (рис. 1.3).

Якщо ε-околиця об'єкта p містить більше точок ніж MinPts, то створюється новий кластер з кореневим об'єктом p. Потім DBSCAN ітеративно збирає об'єкти безпосередньо щільно-досяжні з кореневих об'єктів, які можуть привести до об'єднання кількох щільно-досяжних кластерів. Процес завершується, коли ні до одного кластеру не може бути додано жодного нового об'єкта. Хоча, на відміну від методів розбиття, DBSCAN не вимагає заздалегідь вказувати число одержуваних кластерів, виникне потреба у вказівках значень параметрів ε і MinPts, які безпосередньо впливають на результат кластеризації. Оптимальні значення

цих параметрів складно визначити, особливо для багатовимірних просторів даних.

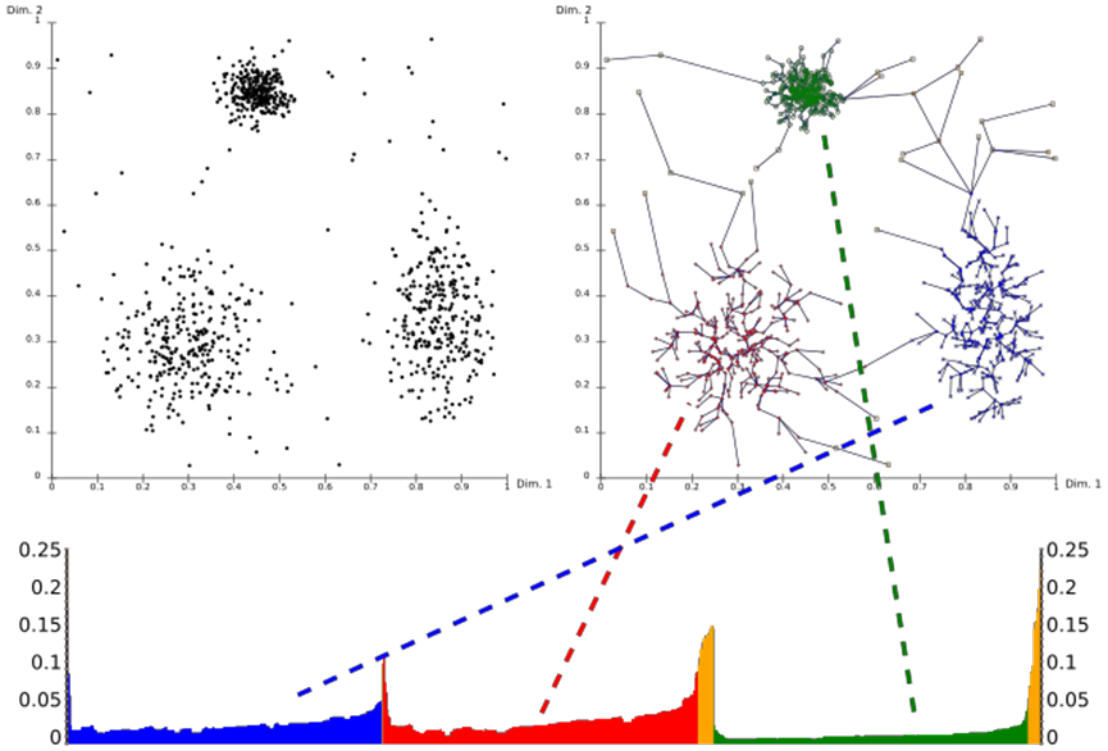


Рис. 1.3. Графік досяжності OPTICS [5]

Алгоритм OPTICS

OPTICS (Ordering points to identify the clustering structure) – це алгоритм знаходження щільності на основі кластерів просторових даних. Його основна ідея схожа на DBSCAN, але він вирішує однин з основних недоліків DBSCAN – проблему визначення значущих кластерів в наборах даних різної щільності. Для цього об'єкти набору даних повинні бути впорядковані (за лінійний час) так, що об'єкти, які просторово близькі, будуть сусідами в упорядкуванні. Крім того, особлива відстань зберігається для кожної точки, яка являє собою щільність, яка повинна бути прийнятна для кластера, щоб обидві сусідні точки належали до тієї ж групи. Для кращого розуміння наведемо графік досяжності OPTICS на рис. 1.4.

Використовуючи графік досяжності, отримуємо ієрархічну структуру

кластерів. Це 2D графік, з упорядкуванням точок, оброблених OPTICS на осі *Ox*, а відстань досяжності на осі *Oy*. Так як точки, що належать до одного кластера, мають низьку досяжність для найближчих сусідів, кластери показують ділянки досяжності. Чим глибша ділянка, тим щільніший кластер.

Зображення на рис. 1.4 ілюструє цю концепцію. У верхній лівій частині показано штучний набір даних. Верхня права частина візуалізує сполучне дерево, створене за допомогою OPTICS, і нижня частина показує ділянку досяжності, обчислену OPTICS. Виявлення кластерів з допомогою цього графіку може бути зроблене вручну, вибравши діапазон на осі *Ox* після візуального огляду та вибравши поріг на осі *Oy* або за допомогою різних алгоритмів, які намагаються виявити ділянки, використовуючи крутизну та локальні максимуми. Кластери, отримані таким чином, можуть мати ієрархічну структуру, чого не може бути досягнуто за допомогою DBSCAN.

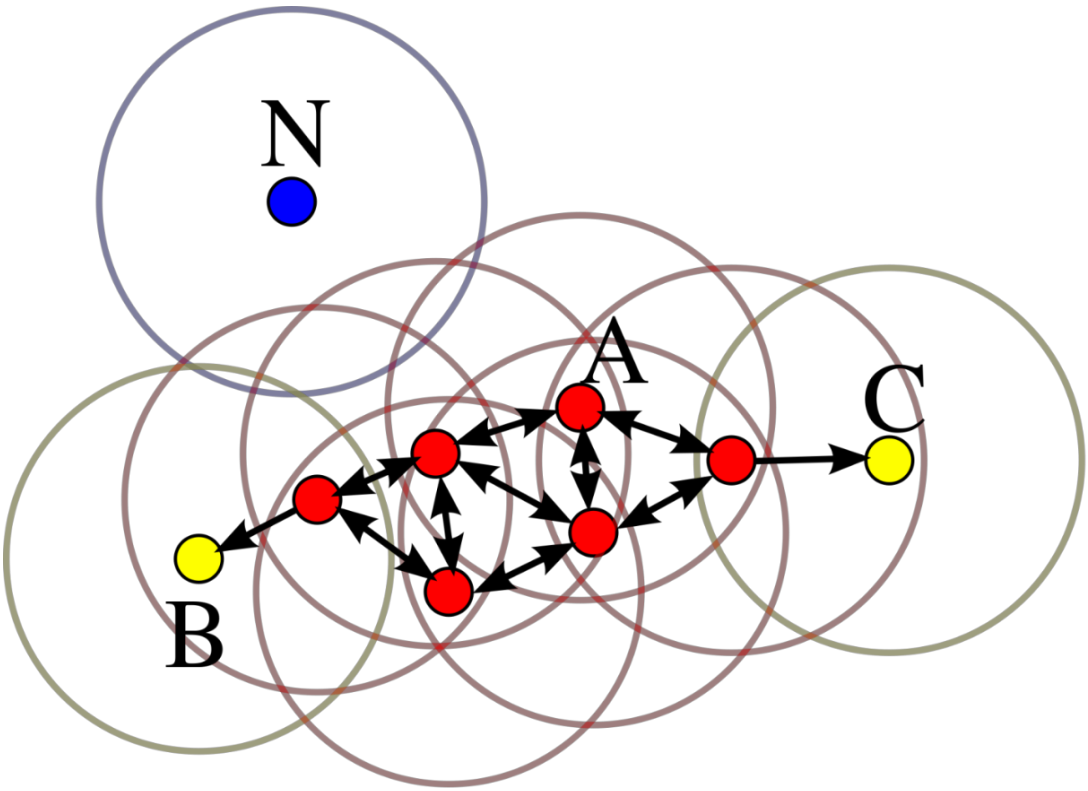


Рис. 1.4. Алгоритм DBSCAN [6]

1.3. Висновки до розділу 1

Таким чином, були проаналізовані ієрархічні, щільнісні та ітеративні методи кластеризації, виявлені переваги та недоліки цих методів.

Для даної задачі був виділений алгоритм kNN, що відноситься до ітеративних методів кластеризації, так як він простий у реалізації та має найбільшу точність на великих об’ємах даних.

2. ФОРМУЛЮВАННЯ ЗАПРОПОНОВАНОГО ПРОГРАМНОГО МЕТОДУ ДЛЯ АНАЛІЗУ РУЗУЛЬТАТІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Аргументація вибору методу кластеризації

В результаті розгляду методів кластеризації виявлено, що їх можна поділити на 3 категорії:

- ітеративні;
- ієрархічні;
- щільнісні.

За основними вимогами вимогами:

- швидкість;
- простота;
- точність.

Проаналізовано три методи кластеризації, які можна використовувати для вирішення даної задачі: метод Байєса (Naive Bayes), метод опорних векторів (Support Vector Machines) та метод *k*-найближчих сусідів (kNN-algorithm) [7]. Нижче наведена таблиця порівнянь цих методів.

Таблиця 1

Порівняння методів кластеризації

Назва методу	Простота реалізації	Точність кластеризації	Швидкість виконання
Метод Байєса	–	+/–	+
Метод опорних векторів	–	+	+
Метод *k*-найближчих сусідів	+	+	–

Так як основною задачею роботи являється атоматизація процесу аналізу результатів тестування програмного забезпечення, що означає

зменшення втручання людини у цей процес, основними вимогами для обрання алгоритму були точність, простота. Час роботи алгоритму не розглядався як основна метрика для вибору алгоритму, так як використання будь-якого з цих алгоритмів загалом значно зменшить час аналізу тестування. Саме тому за основу був вибраний алгоритм *k*-найближчих сусідів (kNN-algorithm), так як він має найпростішу реалізацію та має найбільшу точність на великих об’ємах даних.

На першому кроці алгоритму слід задати число k – кількість найближчих сусідів. На другому кроці знаходяться k записів з мінімальною відстанню до вектора ознак нового об'єкта (пошук сусідів). Функція для розрахунку відстані повинна відповідати наступним правилам:

- $d(x, y) \geq 0$, $d(x, y) = 0$ тоді і тільки тоді, коли $x = y$;
- $d(x, y) = d(y, x)$;
- $d(x, z) \leq d(x, y) + d(y, z)$, за умовою, що точки x, y, z не лежать на одній прямій, де x, y, z – вектори ознак об'єктів, що порівнюються.

Для впорядкованих значень атрибутів Евклідова відстань:

$$D_E = \sqrt{\sum_i^n (x_i - y_i)^2}, \tag{6}$$

де n – кількість атрибутів.

Для рядкових змінних, які не можуть бути впорядковані, може бути застосована функція відмінності, яка задається наступним чином:

$$dd(x, y) = \begin{cases} 0, x = y; \\ 1, x \neq y. \end{cases} \tag{7}$$

При знаходженні відстані враховують значимість атрибутів, яка визначається експертом або аналітиком суб'єктивно, покладаючись на власний досвід. У такому випадку при знаходженні відстані кожен i-й квадрат різниці в сумі множиться на коефіцієнт Z_i. Наприклад, якщо атрибут A в три рази важливіший за атрибут B ($Z_A = 3$, $Z_B = 1$), то відстань буде:

$$D_E = \sqrt{3(x_A - y_A)^2 + (x_B - y_B)^2}. \tag{8}$$

2.2. Застосування алгоритму kNN для поставленої задачі

Для роботи алгоритму для початку потрібно визначитися з тим, які вхідні дані будуть використовуватися. Так як будуть аналізуватися результати виконання автоматизованих тестів, то результатом виконання цих тестів буде траса стеку (stack trace) [9], що буде містити у собі повідомлення про те, що пішло не так у тесті.

У роботі будуть використовуватися результати роботи бібліотеки Junit для мови Java, тому траса стеку буде мати наступний вид, що показаний на рис. 2.1.

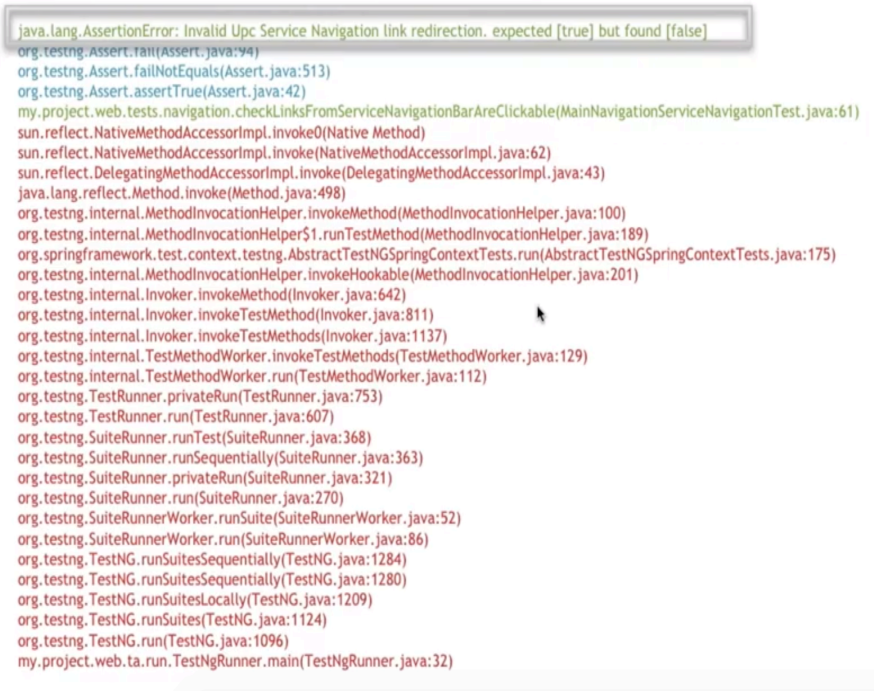


Рис. 2.1. Траса стеку Java

У першому рядку знаходиться повідомлення про причину "падіння" тесту. На наступних 4 рядках відображена інформація про файли, в яких виникла помилка, а далі йдуть помилки, що спливають на всіх рівнях абстракції бібліотеки, що використовується и зазвичай вони ідентичні у всіх автоматизованих тестах.

Так як основна частина цього тесту ідентична для кожного тесту, подавати на вхід алгоритму всю трасу стеку не має сенсу, так я це не принесе

бажаного результату, тому першим кроком буде видалення цього повторюваного тексту.

Після цього залишиться основна інформація, яка унікальна для кожного результату тесту. Проте цього також буде недостатньо, так як залишились ще так звані "шуми" які потрібно прибрати.

```
2019-09-02 07:55:37

java.lang.AssertionError: Invalid Upc Service Navigation
link redirection. expected [true] but found [false]
org.testng.Assert.fail(Assert.java:94)
org.testng.Assert.failNotEquals(Assert.java:513)
org.testng.Assert.assertTrue(Assert.java:42)
my.project.tests.checkLinksAreClickable(MainTest.java:61)
```

Рис. 2.2. Траса стеку з шумами

Наступним кроком буде видалення дати, яка також буде заважати проводити кластеризацію, потім зведення всього тексту до нижнього регістру та видалення знаків пунктуації. Після цього отримуємо текст, з яким можна працювати.

```
java lang assertionError  invalid upc service navigation
link redirection  expected  true      found  false
org testng assert fail assert java
org testng assert failnotequals assert java
org testng assert asserttrue assert java
my project tests checkLinksareclickable maintest java
```

Build #1 Build #2 Build #3 Build #4 Build #5

Рис. 2.3. Текст, готовий до кластеризації

Після того, як збереться багато тексту, що повторюється час від часу, потрібно зібрати частотні покажчики, тобто як часто конкретні слова повторюються у певній категорії. І чим більше тестів, тим більше буде

збиратися тексту з яким можна працювати и таким чином буде зростати точність роботи алгоритму.

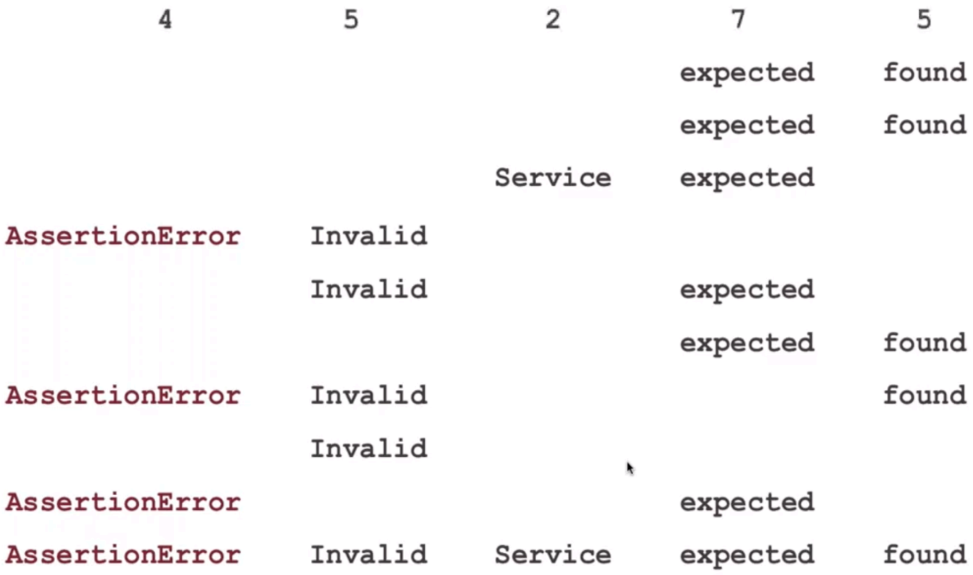


Рис. 2.4. Приклад частотних покажчиків

Після того, як ми маємо всі ці метрики, ми переходимо до частотної характеристики, це саме та частина, де відбувається машинне навчання і називається вона TF-IDF метрикою. В основному ця метрика показує, як часто це слово з'являється у документі і наскільки важливе це слово серед всіх документів, що знаходяться у бібліотеці.

TF (term frequency – частота слова) – відношення числа входжень обраного слова до загальної кількості слів документа. Таким чином, оцінюється важливість слова *t* в межах обраного документа.

$$TF = \frac{n_i}{\sum_k n_k}, \tag{9}$$

де n_i є число входжень слова в документ, а в знаменнику – загальна кількість слів в документі.

IDF (inverse document frequency – обернена частота документа) – інверсія частоти, з якою слово зустрічається в документах колекції. Використання IDF зменшує вагу широковживаних слів.

$$\text{IDF} = \log \frac{|D|}{|d_i \supset t_i|}, \tag{10}$$

де:

– |D| – кількість документів колекції;
– $|d_i \supset t_i|$ – кількість документів, в яких зустрічається слово t_i (коли $n_i \neq 0$).

Вибір основи логарифму у формулі не має значення, адже зміна основи призведе до зміни ваги кожного слова на постійний множник, тобто вагове співвідношення залишиться незмінним. Іншими словами, показник TF-IDF це добуток двох множників: TF та IDF.

Ця частотна характеристика допоможе визначити, наскільки слово важливе для конкретного документу і на скільки це слово важливе у категоризації конкретного типу документу.

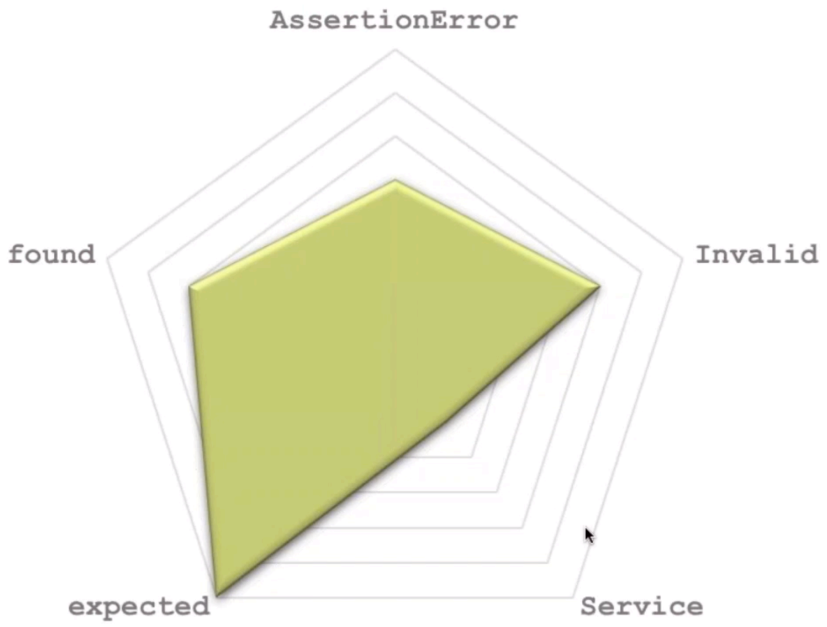


Рис. 2.5. Графічне представлення метрики TF-IDF

Ми маємо всі ці слова, які відносяться до різних категорій, наприклад, таких як Product Bug, System Issue або Automation Issue, і ця категоризація буде проводитися людиною на початку, щоб машинне навчання розуміло, як цей документ, тобто цей стектрейс, відноситься до конкретної причини "провалу" тесту.

Зрештою, ця інформація буде представлена у векторному вигляді.

Тобто аналіз тексту за допомогою алгоритмів машинного навчання означає конвертування всього цього великого тексту у математичний вектор і цей вектор буде розташований у багатовимірному просторі, де кожен простір буде відноситися до певного слова.

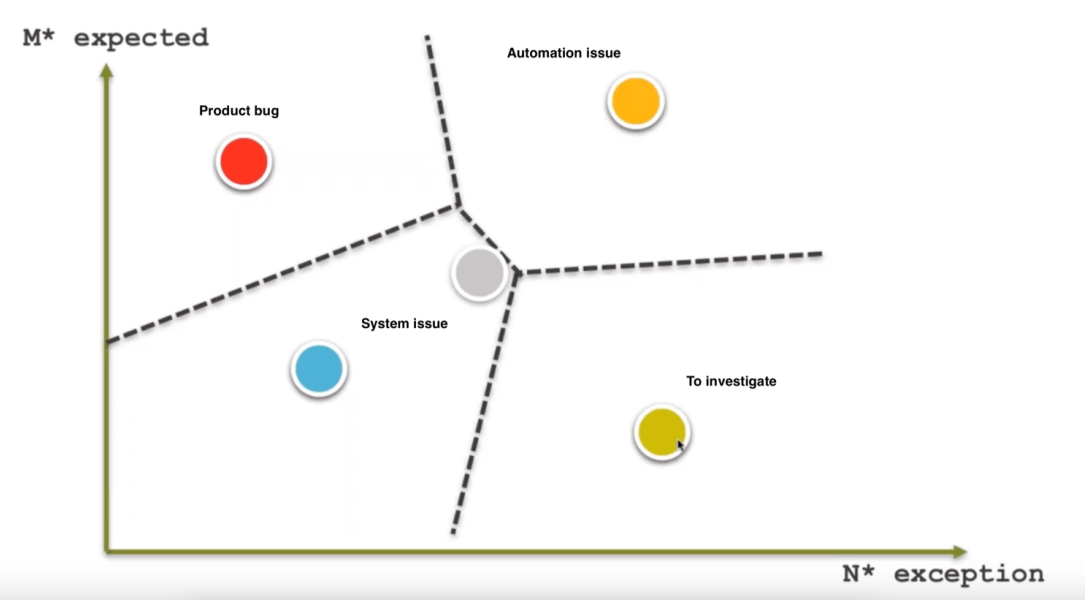


Рис. 2.6. Двовимірний приклад кластеризації результатів тестування

2.3. Висновки до розділу 2

В даному розділі було проведено ґрунтовну роботу в напрямку розроблення методу для аналізу результатів тестування програмного забезпечення. Було виконане наступне:

– вибір базового методу;

– опис реалізації вибраного методу.

Проаналізовано три методи кластеризації, які можна використовувати для вирішення даної задачі кластеризації результатів виконання тестування програмного забезпечення, а саме метод Байєса, метод опорних векторів та метод *k*-найближчих сусідів.

Показано, що метод *k*-найближчих сусідів має найпростішу реалізацію та має найбільшу точність на великих об’ємах даних для аналізу результатів тестування програмного забезпечення.

3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МЕТОДУ

3.1. Вимоги до програмного забезпечення

Система повинна надавати можливість:

- регістрації та входу з платформи GitHub;
- переглядати виконання тестів у режимі реального часу;
- автоматичної кластеризації результатів тестування;
- задавати категорії та підкатегорії для тестів, що закінчилися з помилкою;
- зберігати до 10 запусків тесту;
- об’єднувати декілька запусків в один;
- прикріпляти зображення, частини коду до конкретного результату тесту;
- редагувати категорії та підкатегорії тестів, що закінчилися з помилкою;
- фільтрувати результати виконання тестів за категоріями та підкатегоріями;
- додавати, змінювати, видаляти опис результатів тустування.
- налаштовувати параметри для автоаналізу;
- відображати результати в графічному вигляді.

3.2. Архітектура розробленого програмного забезпечення

Архітектура програмної системи була реалізована у вигляді мікросервісів. На відміну від монолітної архітектури, мікросервісна має більше переваг.

Мікросервіси – це різновид сервіс-орієнтованої архітектури (SOA), що застосовується для формування розподілених програмних систем. Модулі в мікросервісной архітектурі взаємодіють у мережі, виконуючи при цьому єдину мету. На даний момент мікросервіси поступово витісняють монолітні додатки і перетворюються в стандарт розвитку програмних систем.

Важливо розуміти, що під сервісом розуміється цілий набір послуг і визначена функціональність для користувача. Мікросервіси – це дроблення функціональності так, щоб вона була доступна іншим частинам системи. Поділ функціональності настільки дрібний, що всередині кожного мікросервіса реалізовано дуже небагато функцій. Мікросервісна архітектура – це кілька функціональних модулів, що взаємодіють через мережу.

Головна відмінність мікросервісов від моноліту – у використанні спеціалізованих простіших програм (модулів) при виконанні сценарію програми.

І, що найзручніше, модулі можуть знаходитися на різних серверах і їх взаємодія відбувається через мережу за протоколонезалежною технологією.

Мікросервісна архітектура має багато переваг:

- симетрична архітектура (в монолітних додатках – ієрархічна);
- взаємозамінність мікросервісів;
- незалежність мікросервісів один від одного;
- організація модулів навколо окремих функцій;
- написання мікросервісів за допомогою будь-яких програмних засобів, оптимальних для кожного конкретного модуля, при цьому вони добре "розуміють" один одного завдяки інтерфейсу;
- мікросервіси викликаються тільки користувачем, але не один одним.

Саме тому була обрана мікросервісна архітектура, що наведена на рис. 3.1.

Вся система ділиться на дві частини – клієнтську та серверну. До клієнтської частини входять такі сервіси, як Logger, Agent, Client.

Client – це інтеграції API. HTTP-клієнти, які обробляють надсилання HTTP-запиту.

Agent – це інтеграція фреймфорку. Спеціальні репортери / слухачі, які відстежують тестові події та викликають надсилання подій через клієнта.

Logger – це інтеграція логів, які допомагають збирати журнали, пов'язують його з тестовим кодом через агент та надсилають на сервер через клієнта.

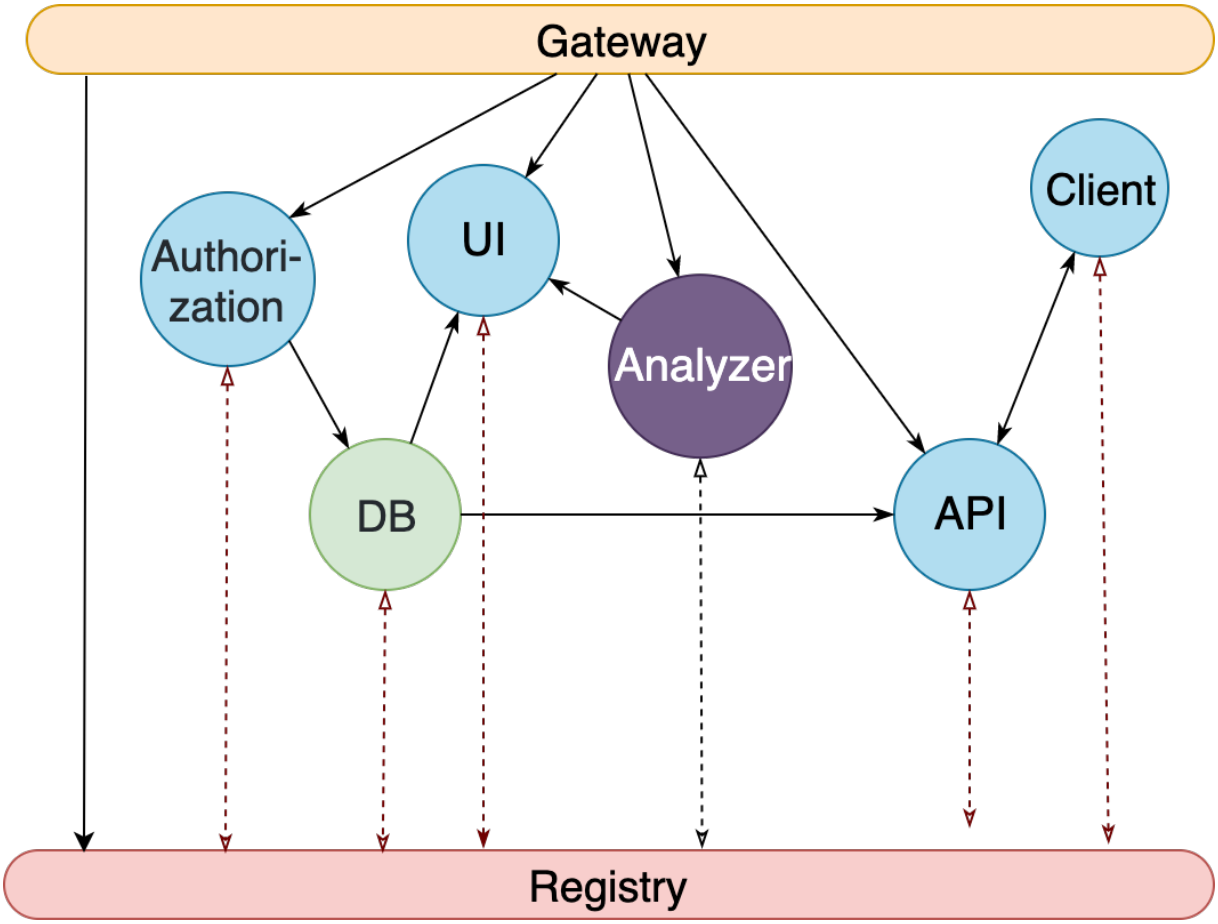


Рис. 3.1. Архітектура програмного забезпеченя

Gateway – є головною точкою входу до служб прикладних програм. Цей сервіс відповідає за маршрутизацію запитів на відповідне обслуговування та балансування навантаження. Шлюз зв’язується з реєстром послуг, щоб отримати список фактичних служб, яким дозволено маршрутизувати трафік.

Registry – це інструмент, який зберігає фактичний список запущених служб із доданою метаінформацією. Він здійснює перевірку стану для кожної запущеної служби, щоб забезпечити доступність.

API – це серце системи. Він відповідає за обробку вхідних запитів агентів та інтерфейсу користувача.

Authorization – це модуль, який авторизує користувачів та створює / відкликає токени користувачів. Він підтримує різні типи механізмів аутентифікації:

- Basic Auth;
- GitHub Auth (OAuth2);
- LDAP Auth – це сервер OAuth2, який авторизує користувача за допомогою механізмів, згаданих нижче, і створює внутрішній токен OAuth2, який використовується інтерфейсом користувача та агентами. Існує два типи токенів:
  - UI (токен, що закінчується);
  - API – токен, що не закінчується, призначений для використання на стороні агента.

Analyzer – зберігає індекс журналів користувачів проекту і надає можливість здійснювати пошук за цим індексом. Використовується функцією автоматичного аналізу.

UI – сервіс, що відповідає за клієнтську частину системи.

DB – сервіс, що відповідає за базу даних системи.

3.3. Особливості програмної реалізації застосунку

3.3.1. *Серверна частина*

Для реалізації основної частини системи було вибрано мову програмування Java. Вона є статично-типізованою мовою програмування загального призначення, це об'єктно-орієнтована мова. Java розроблена за принципом WORA (написати один раз та запускати будь-де), вона була розроблена для роботи на будь-якій платформі та з якомога меншими залежностями за допомогою віртуальної машини Java (JVM).

Переваги Java

Об'єктивно-орієнтована: у Java все є об'єктом. Додаток може бути легко розширений, так як він заснований на об'єктній моделі.

Платформонезалежна: на відміну від багатьох інших мов, включаючи C і C ++, Java, коли була створена, вона не компілювалась у платформі конкретної машини, а в незалежному від платформи байт-коді. Цей код розповсюджується через Інтернет і інтерпретується у віртуальній машині Java (JVM), на якій він працює в даний момент.

Проста: процеси вивчення і введення в мову програмування Java залишаються простими. Якщо розуміти основні концепції об'єктно-орієнтованого програмування, то вона буде простою в освоєнні.

Безпечна: методи перевірки автентичності засновані на шифруванні з відкритим ключем.

Архітектурно-нейтральна: компілятор генерує архітектурно-нейтральні об'єкти формату файлу, що робить скомпільований код виконуваним на багатьох процесорах, з наявністю системі Java Runtime.

Портативна: архітектурно-нейтральна і не має значної залежності від реалізації аспектів специфікацій – все це робить Java портативною. Компілятор в Java написаний на ANSIC з повною переносимістю, що є підмножиною POSIX.

Багатопотокова: функції багатопоточності, можна писати програми, які можуть виконувати безліч завдань одночасно. Введення в мову Java цієї конструктивної особливості дозволяє розробникам створювати налагоджені інтерактивні додатки.

Інтерпретована: Java байт-код перетворюється "на льоту" в машинні інструкції та ніде не зберігається. Це робить процес більш швидким і аналітичним, оскільки відбувається додаткове з'вязування з невеликою вагою процесу.

Високопродуктивна: введення Just-In-Time компілятора дозволило отримати високу продуктивність.

Динамічна: програмування на Java вважається більш динамічним, ніж на C або C ++, так як вона передбачена для адаптації до зміни умов. Програми можуть виконати велику кількість під час обробки інформації, яка може бути використана для перевірки та вирішення доступу до об'єктів на час виконання.

3.3.2. *Аналіз баз даних, що відповідають визначеним вимогам*

Cassandra

Cassandra – це високо-масштабоване, узгоджене (автоматично відновлює узгодженість даних), розподілене, засноване на структурі ключ-значення сховище. Cassandra пропонує разом технології розподілених систем Dynamo і модель даних Google BigTable [12].

Однією з найсильніших сторін цієї бази даних є можливість обробляти величезну кількість неструктурованих даних. У випадках, коли база даних потребує швидкого масштабування з мінімальним збільшенням адміністративної роботи, Cassandra може стати гарним вибором.

Cassandra може обробляти завантаження програм, таких як Instagram, які щоденно завантажують приблизно 80 мільйонів фотографій у базу даних.

Cassandra використовує рядки та стовпці і дозволяє змінювати їх назву та формат. Вона використовує суміш табличного та ключового значення. На відміну від типової системи керування реляційною базою даних (РСУБД) [13], таблиці можна створювати, змінювати та скидати під час роботи бази даних та обробки запитів.

Сімейста стовпців схожі на таблиці в РСУБД та містять рядки та стовпці, причому кожен рядок має унікальний ключ. На відміну від традиційної СУБД, всі рядки в таблиці не повинні мати однакові стовпці. Ці стовпці також можуть бути додані "на льоту" [14]. Отримати доступ до них за допомогою мови запиту Cassandra (CQL). Хоча CQL подібний до

синтаксису SQL, Cassandra не є реляційною базою даних, тому вона має різні способи зберігання та отримання даних.

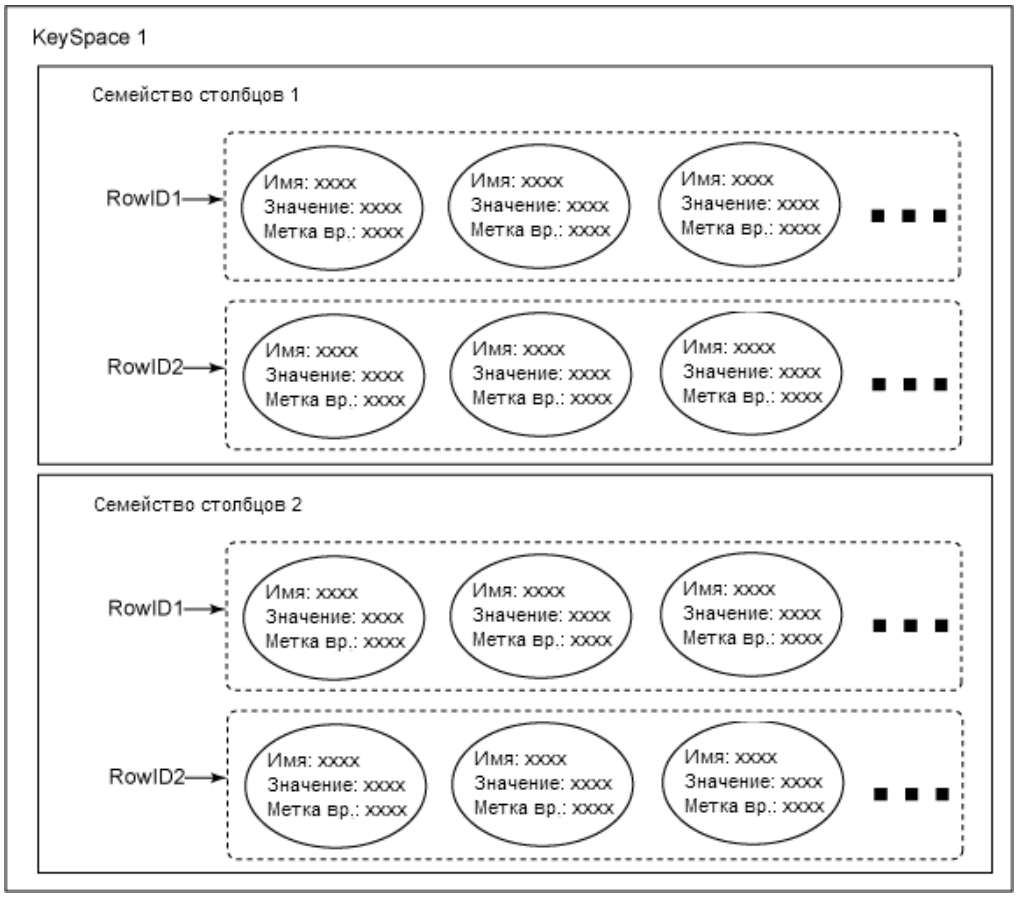


Рис. 3.2. Модель даних СУБД Cassandra

Cassandra дозволяє робити реплікацію даних з "коробки". Потрібно лише вказати кількість вузлів, у які вона повинна скопіювати дані і вона виконає решту процесу.

Переваги СУБД Cassandra:

1. Висока масштабованість і надійність без елементів, відмова яких призводить до виходу з ладу всієї системи.
2. Реалізація сімейства NoSQL Column.
3. Дуже висока пропускна здатність для операцій запису і хороша пропускна здатність для операцій зчитування.
4. SQL-подібна мова запитів (починаючи з версії 0.8) і підтримка пошуку за допомогою вторинних індексів.

5. Підтримка реплікації.
6. Гнучка схема.

Недоліки СУБД Cassandra:

1. Погано реалізовано пошук по діапазону.
2. Дуже багато налаштувань винесені на рівень кластеру.

MongoDB

На відміну від реляційних баз даних, MongoDB являє собою документо-орієнтовану модель даних, завдяки чому набагато швидше працює, краще масштабується та проста у використанні [15].

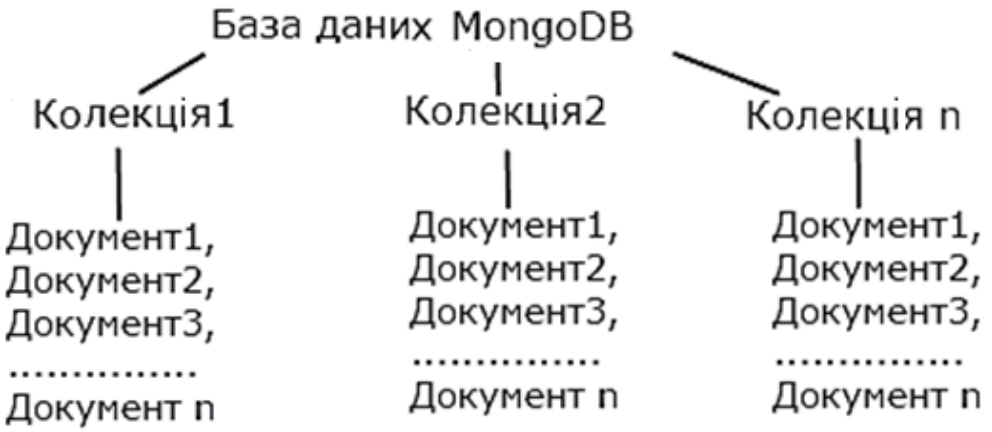


Рис. 3.3. Модель даних MongoDB

Одним із найпопулярніших стандартів для обміну даними та їх зберігання є JSON (JavaScript Object Notation). JSON доволі ефективно дозволяє описати складні за структурою дані. Цей спосіб зберігання даних доволі схожий на спосіб зберігання в MongoDB, хоча насправді JSON не використовується. Для зберігання даних в MongoDB використовується формат, що називається BSON (скорочення від binary JSON) [16].

MongoDB використовує свою мову запитів, що дозволяє отримати доступ до збережених даних. Оскільки він не має схеми, можна створювати документи, при цьому не задавати їм якусь певну структуру з самого початку, як це потрібно робити у реляційних базах даних [17].

Переваги MongoDB:

1. Підтримка індексації.
2. Дозволяє балансувати навантаження.
3. Можливість використовувати в якості файлового сховища.
4. Підтримка агрегації відповідно до парадигми MapReduce.
5. Підтримка асинхронної реплікації.

Недоліки MongoDB:

1. Проблеми з узгодженістю.
2. Блокування на рівні документу.
3. Деякі проблеми з масштабуванням (якщо об’єм даних перевищує 100ТБ).

CouchDB

CouchDB використовує документоване сховище з даними, що представлені у форматі JSON [18]. Він пропонує RESTful HTTP API для читання, додавання, редагування та видалення документів бази даних. Кожен документ складається з полів та вкладень. Поля можуть складатися з цифр, тексту, логічних значень, списків тощо.

Перегляди в CouchDB аналогічні індексам у SQL. Їх можна використовувати для фільтрації документів, отримання даних у певному порядку та створення ефективних індексів для пошуку документів в їх межах. Після того, як отримано індекси, вони можуть відображати зв’язки між документами. Ці результати перегляду зберігаються в структурі індексу B-tree.

CouchDB використовує MapReduce [19], таких двоетапний процес, який переглядає всі документи та в результаті створює масив значень, що складається з упорядкованого списку пар ключ / значення. Відображення відбувається один раз після створення документа. Після цього він не змінюється, поки документ не буде оновлено.

CouchDB був написаний в Erlang і доступний для Android, BSD, iOS,

Linux, OS X, Solaris та Windows.

Він має підтримку багатьох мов програмування, зокрема: Haskell, Java, C, C #, ColdFusion, Erlang, Objective-C, OCaml, Perl, PHP, PL / SQL, Python, JavaScript, Lisp, Lua, Ruby і Smalltalk.

CouchDB підтримує як master-master, так і master-slave реплікацію [20]. Це забезпечує низький час затримки доступ до даних незалежно від їх місцезнаходження. Реплікація в CouchDB настільки ж проста, як надсилання HTTP-запитів у базу даних.

CouchDB надсилає будь-які зміни, що відбуваються в джерелі, до цільової бази даних. Це односпрямований процес. Якщо потрібен двонаправлений процес, доведеться ініціювати реплікацію на цільовому сервері, коли вона є джерелом, а віддалений сервер є призначенням.

Обрання бази даних

На попередньому етапі був проведений аналіз можливостей кожної з баз даних, що найбільш відповідають зазначеним до них вимогам. Отримані результати аналізу наведено в табл. 2.

Таблиця 2

Порівняння баз даних

СУБД	Переваги СУБД	Недоліки СУБД
Cassandra	Висока масштабованість, SQL-подібна мова запитів	Погано реалізовано пошук по діапазону, дуже багато налаштувань винесені на рівень кластеру.

Продовження табл. 2

MongoDB	Балансування навантаження, можливість використовувати в якості файлового сховища, підтримка асинхронної реплікації	Проблеми з узгодженістю, блокування на рівні документу
CouchDB	REST запити з "коробки"	Не призначений для часто оновлюваних даних, не має вбудованого повнотекстового пошуку

У якості СУБД для зберігання даних було прийняте рішення обрати MongoDB, так як вона більше підходить для часткової зміни даних, має вбудований повнотекстовій пошук.

3.2.3. *Клієнтська частина*

Фреймворк Angular

Angular – JavaScript-фреймворк з відкритим програмним кодом, що розробляється компанією Google. Призначений для створення динамічних веб-додатків. Angular написаний на мові TypeScript (JavaScript зі строгою типізацією) [1]. Він реалізує основні та додаткові функції як набір бібліотек, що імпортуються у додаток і використовує архітектуру показану на рис. 3.4.

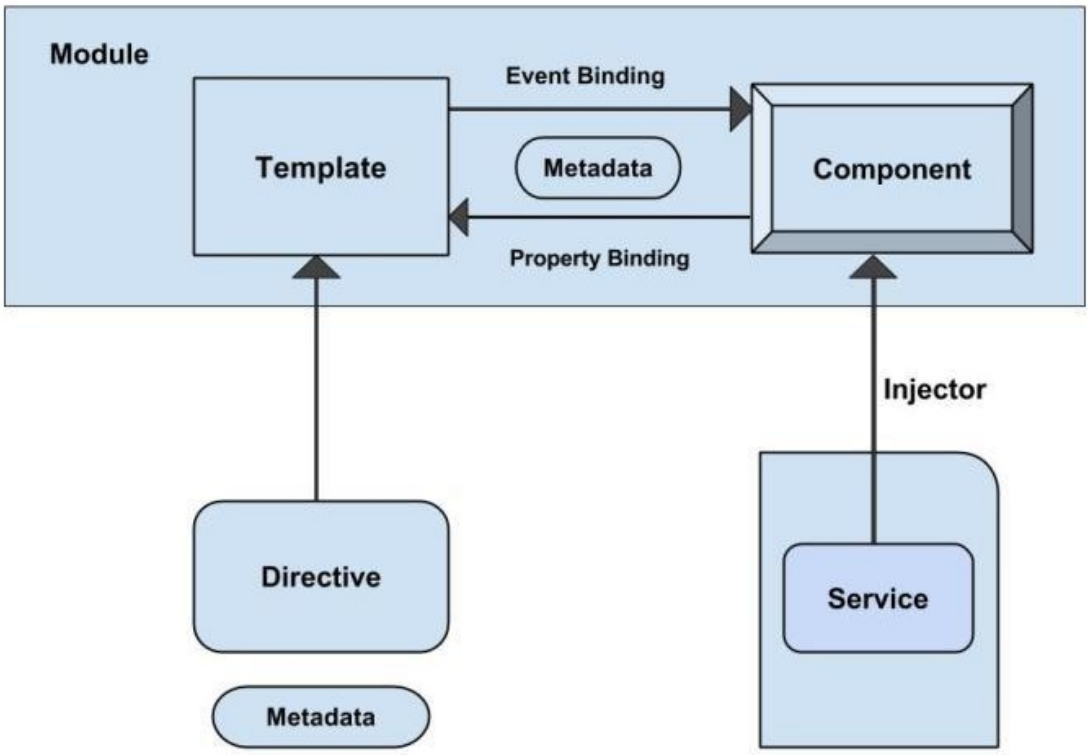


Рис. 3.4. Архітектура фреймворку Angular

Основними "будівельними" блоками Angular є NgModules, що надають контекст компіляції для компонентів. NgModules збирають відповідний код у функціональні набори. Додаток написаний на Angular визначається набором NgModules, він має принаймні кореневий модуль, який дозволяє завантажувальний файл, і зазвичай має багато інших функціональних модулів.

Компоненти визначають представлення (views), які є набором елементів, що відображаються у вікні браузера. Angular може вибрати і змінювати їх відповідно до логіки програми та даних. Кожен додаток має принаймні кореневий компонент.

Компоненти користуються службами (services), які забезпечують особливі функції, не пов'язані безпосередньо з представленнями. Служби можуть бути введені в компоненти як залежності, що робить код модульним, багаторазовим і ефективним.

Метадані класу компонентів пов'язують його з шаблоном, який визначає вид. Шаблон об'єднує звичайний HTML з директивами Angular та обов'язкову розмітку, що дозволяє Angular змінювати HTML перед тим, як виводити його на екран.

Компоненти програми зазвичай мають багато представлень, розташованих ієрархічно. Angular використовує маршрутизатор (Router), який дозволяє визначити шляхи навігації для представлень.

Angular використовує двостороннє зв’язування даних (two-way data binding). Шаблон об'єднує HTML розміткою Angular, що дозволяє змінювати елементи HTML перед їх показом. Шаблонні директиви забезпечують логіку програми, а обов'язкова розмітка з'єднує дані програми та об'єктну модель документа (DOM).

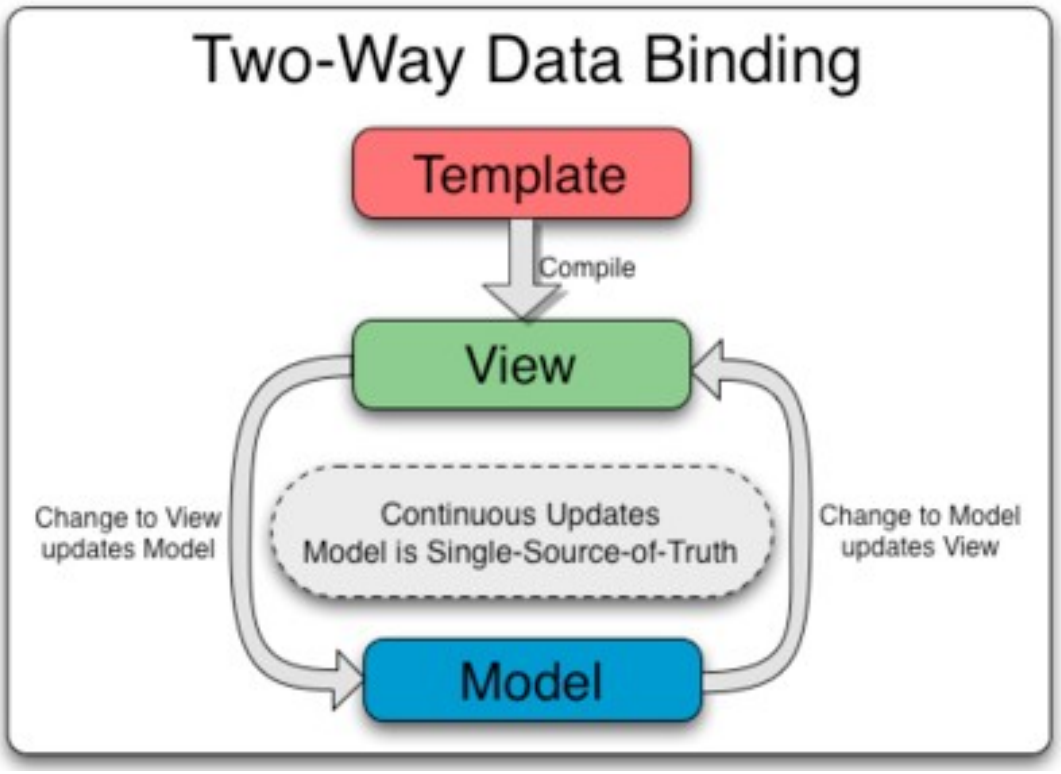


Рис. 3.5. Двостороннє зв’язування даних

Особливості фреймворку:

1. Створення не тільки нових тегів для HTML-розмітки, але й редагування існуючих.

2. Винесення верстки в окремий файл.
3. Налаштування поведінки компонента шляхом зміни параметрів роботи компонента CSS-класами.
4. Можливість вказувати, які параметри можуть прийти до компонента на вхід і які події можуть від нього відходити.

Бібліотека ReactJS

ReactJS – відкрита JavaScript бібліотека для створення користувацьких інтерфейсів, що створена вирішувати такі проблеми, як часткове оновлення вмісту веб-сторінки, що виникають в процесі створення односторінкових додатків [2]. React – мова шаблонів, що в поєднанні з декількома функціями дозволяє відображати HTML, тобто результатом роботи є HTML.

ReactJS використовує концепції реактивного програмування: зміна результату роботи залежить від стану компонентів. Так, $A = B + C$, і результат A завжди буде залежати від значень B і C.

ReactJs постійно працює з DOM, перемальовуючи його при зміні умов (та частина DOM, яку змінює ReactJs, називається компонентом). Раніше подібна практика сильно б відбилася на продуктивності додатка, але розробники ReactJs підійшли до вирішення даного питання кардинально: вони повністю переписали DOM на Javascript.

Важливою особливістю ReactJs є використання JSX. Це надбудова над JavaScript, що дозволяє використовувати про-XML синтаксис в JavaScript коді. JSX – це своєрідне поєднання JavaScript і HTML, які в зв'язці є незвичним синтаксисом для більшості розробників. Стандартом вважається поділ JS частини від розмітки, що ускладнює спостереження за змінами. JSX дозволяє бачити всі процеси в одному місці, не відволікаючись на складність грамотного і валідного коду. Після компіляції JSX виходить чистий JS.

Переваги ReactJS:

1. Розмір бібліотеки. Очевидно, що даний показник не є

вирішальним, та все ж потрібно визначити той факт, що ядро React + Flux займає в 3 рази менше Angular.

2. Швидкість роботи. Завдяки підходу з віртуальним DOM та JSX значно випереджає конкурентів.
3. Компіляція в HTML на сервері.
4. Існує можливість (і інструменти які її реалізують) видавати повністю автономну html сторінку яка не вимагає повторної компіляції на клієнті (що навіть пришвидшує процес візуалізації). Власне це дозволяє додатку залишатися частково робочим навіть з вимкненим або недоступним JavaScript на клієнті.
5. Сутність JSX.
6. Навіть якщо опустити сам факт, що це більш продумане і відповідно більш швидке (в плані швидкодії роботи) рішення, сама сутність JSX дає ще перевага при обробці помилок. Наприклад, якщо в процесі розробки або злиття гілок, була допущена помилка в шаблоні представлення моделі. Що в цьому випадку відбудеться? Власне нічого – додаток завершить роботу на клієнтській стороні, не відобразивши нічого (це вже як пощастить). Звичайно, якщо додаток покрито повністю тестами, то можливо тести виявлять проблему. Якщо допуститься помилка в JSX – додаток не пройде збірку, видавши при цьому номер рядка, де допущена помилка, а також перелік попереджень, де потенційно може виникнути проблема.

Недоліки ReactJS:

1. Складна документація.
2. На сайті розробника навчальна документація розкидана по різних вкладках, інформація не структурована. Однак якщо з ReactJS працює ціла команда, цю проблему вдається швидко вирішити.
3. Підтримка браузерів.
4. Не всі стандартні браузери підтримують React. Для подолання цієї

проблеми пропонується використовувати додаткові плагіни, наприклад, бібліотеку ES5-shim для підтримки IE8. Існують і інші розширення для ReactJS, однак, з огляду на вагу фреймворка, їх використання краще звести до мінімуму.

5. Більшість віджетів – свої.
6. ReactJS досить молодий фреймворк, тому все, навіть стандартні віджети доведеться прописувати заново.

Бібліотека Vue.js

Vue.js – це JavaScript бібліотека для створення веб-інтерфейсів з використанням шаблону архітектури MVVM (Model-View-ViewModel).

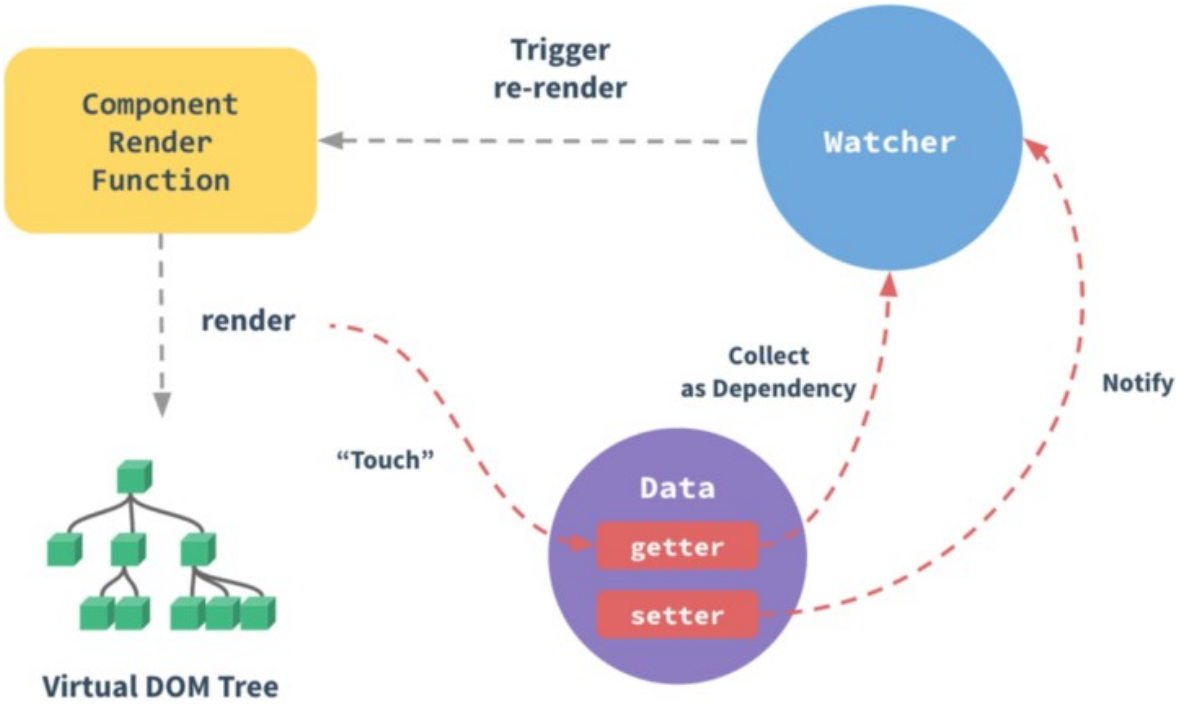


Рис. 3.6. Архітектура Vue.js

Оскільки Vue працює тільки на "рівні представлення" і не використовується для проміжного програмного забезпечення і серверної частини, він може легко інтегруватися з іншими проектами і бібліотеками. Vue.js містить широку функціональність для рівня представлення і може використовуватися для створення потужних односторінкових веб-додатків.

Функції Vue.js:

1. Реактивні інтерфейси.
2. Декларативний рендеринг.
3. Зв'язування даних.
4. Директиви (всі директиви мають префікс "V-". В директиву передається значення стану, а в якості аргументів використовуються HTML атрибути або Vue JS події).
5. Логіка шаблонів.
6. Компоненти.
7. Обробка подій.
8. Переходи і анімація CSS.
9. Фільтри.

Основна бібліотека Vue.js дуже маленька (всього 17 кБ). Це гарантує, що навантаження ваш проект, реалізований за допомогою Vue.js, мінімальне, тобто сайт буде швидко завантажуватися [3].

Vue підходить для невеликих проектів, яким необхідно додати трохи реактивності, уявити форму за допомогою AJAX, відобразити значення при введенні даних користувачем, авторизація або інші аналогічні завдання. Vue легко масштабується і добре підходить для об'ємних проектів, тому його називають прогресивним фреймворком.

Vue також відмінно підходить для великих односторінкових додатків завдяки своїм основним компонентам, таким як Router і Vuex. З Vue можна як використовувати загальнодоступні API для створення додатків, так і реалізовувати додатки, що виконуються сервером. Але Vue найкраще підходить для розроблення рішень, які використовують зовнішні API для обробки даних [5].

За допомогою Vue також можна створювати клієнтську частину блогу на популярних CMS. Vue.js відмінно підходить і для розроблення динамічних інтерфейсів, які адаптуються під користувача.

Обрання технології клієнтської частини

На попередньому етапі вибору технології був проведений аналіз можливостей кожної з технологій було проведено вивчення можливостей кожної з технологій, що найкраще відповідають зазначеним до них вимогам. Отримані результати аналізу наведено в табл. 3.

Таблиця 3

Порівняння технологій для клієнтської частини

Назва технології	Мова програмування	Переваги технології	Недоліки технології
Angular	JavaScript	Містить у собі все, що потрібно для створення повноцінного додатку, дуже легко підтримувати додатки, хороша документація	Досить великий поріг входження, розмір фреймоврку
ReactJS	JavaScript	Розмір бібіотеки, Швидкість роботи за рахунок віртуального DOM, можливість компіляції в HTML на сервері	Складна документація, Так як це лише бібліотека, для розробки повноцінного додатку необхідне встановлення додаткових бібліотек

Продовження табл. 3

Vue.js	JavaScript	Розмір бібіотеки, Швидкість роботи за рахунок віртуального DOM, дуже проста в освоєнні	Компонентний підхід не настільки гнучкий як в ReactJS

Як бачимо з табл. 3, у кожної з технологій є свої як переваги, так і недоліки. Тому, проаналізувавши ці фреймворки та визначивши найбільш підходящий для даного проекту, було прийняте рішення використовувати Angular, адже він досить добре документований, дозволяє легко розпочати роботу над додатком і не потрібно завантажувати додаткові модулі.

3.4. Інтерфейс користувача

3.3.1. *Авторизація*

Перше, що бачить користувач у системі – сторінку логіну, де він може увійти за допомогою github, що дуже зручно, так як йому не потрібно потім налаштовувати профіль и все повідомлення будуть приходити на його пошту, що прив’язана до github.

Вся інформація з виконання тестів знаходиться на головній сторінці, що з’являється після авторизації до системи.

На цій сторінці ми бачимо виконання всіх запусків тестів, де в кожному запуску є його назва, запуску, час коли він був запущений, кількість помилок. що виникли, та відразу результат автоаналізу, тобто кластеризацію помилок за категоріями. Як видно з рис. 3.7, перший тест виконався з 8 помилками, де 6 з них система віднесла до таких категорій, як Product bug, Automation bug, System issue а дві з них залишились у вкладці To investigate, це означає, що користувачу потрібно самому переглянути ці помилки та віднести їх до якоїсь існуючої категорії або створити нову.

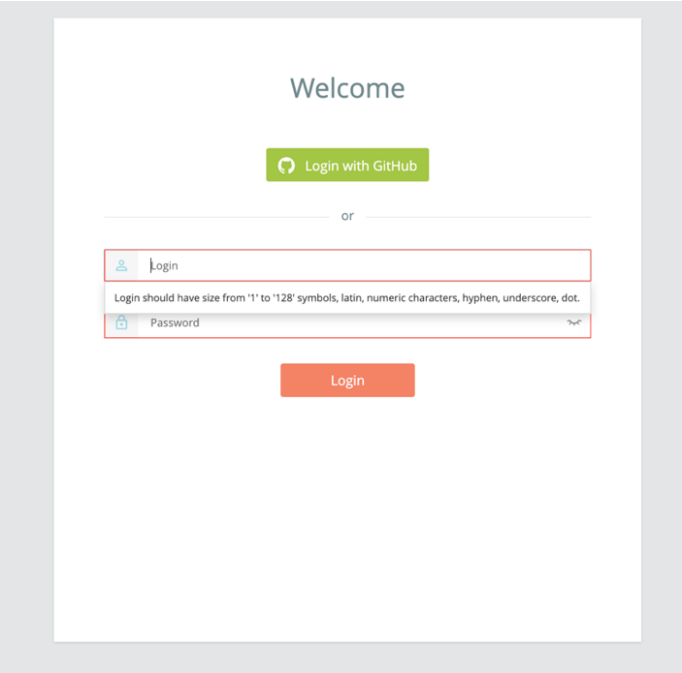


Рис. 3.6. Сторінка авторизації

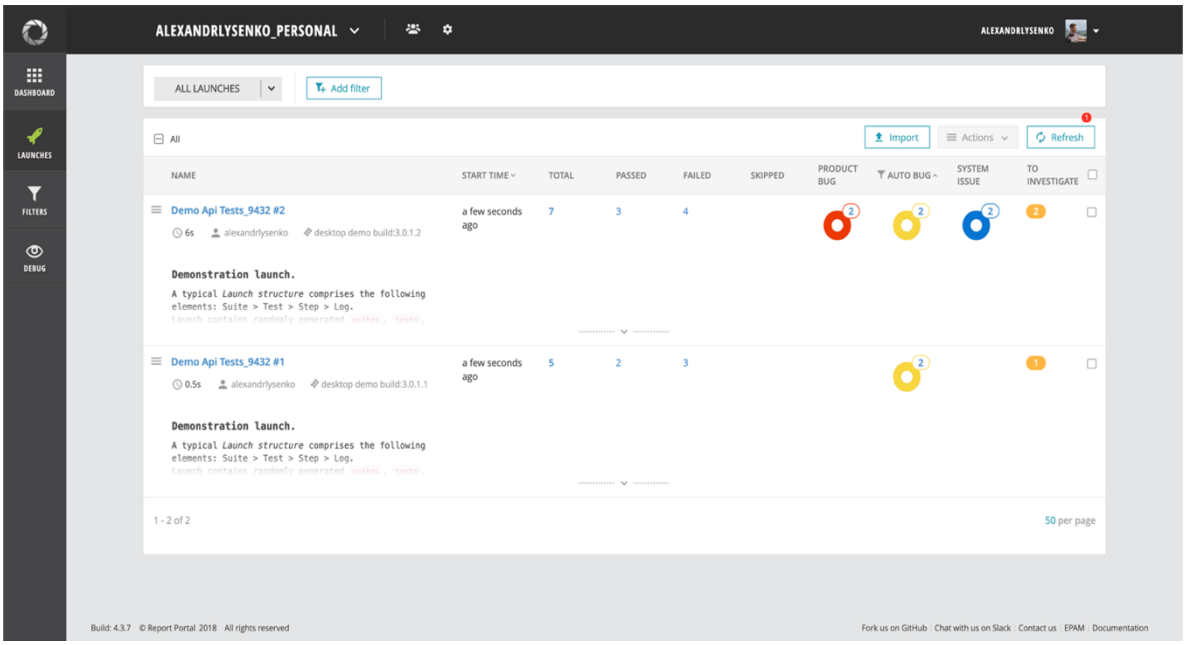


Рис. 3.7. Сторінка запусків

Система має таку структуру рівнів організації тестів:

Launch – Test Suite – Test Case – Test step.

– Launch – містить у собі всі набори тестів що були виконані у запуску;

– Test Suite – це набір тест кейсів, які об'єднані тим що відносяться

до одного тестуючого модулю, функціональності, пріоритетністю або одного типу тестування. Кожен тест сьют складається з більш ніж одного тест кейса і часто виконується всієї "пачкою" в процесі тестування рис. 3.7;

– Test Case – це формально описаний алгоритм тестування програми, спеціально створений для визначення виникнення в програмі певній ситуації, певних вихідних даних;
– Test Step – описують кроки, для того щоб відтворити баг. Кроки рекомендують максимально скорочувати, тобто знайти найкоротший шлях для відтворення помилки і описати в кроках, і дуже важливо, щоб вони залишалися максимально зрозумілими для розробників.

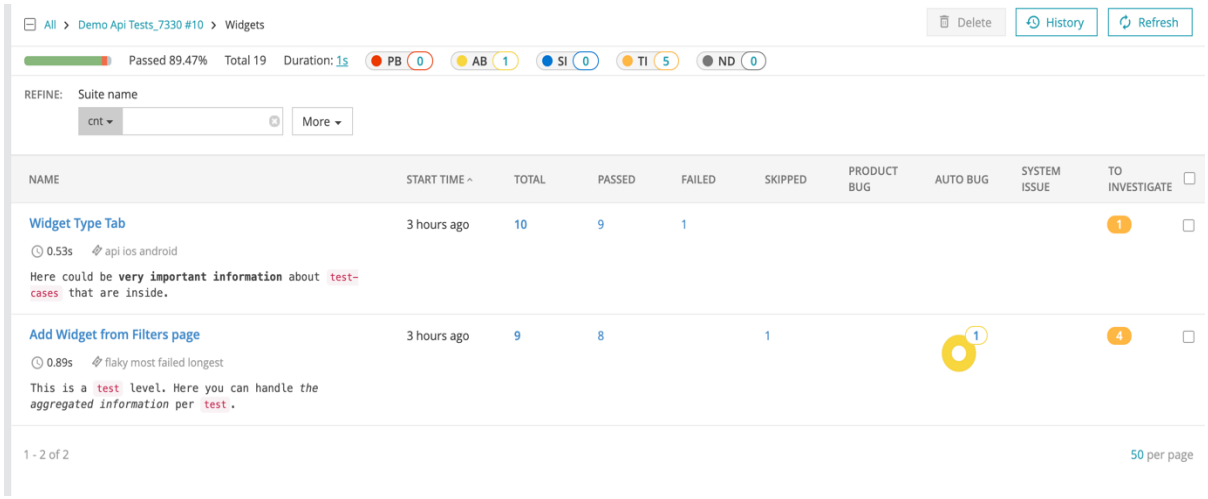


Рис. 3.8. Рівень test suite

Саме на рівні Step у системі можна подивитися, на якому кроці сталася помилка, подивитися історію запуску.

На рис. 3.8 бачимо результат виконання тест-кейсу зі всіма його кроками та помилки, що виникли під час запуску. Зверху знаходиться історія запуску цього тесту, система зберігає останні 10 запусків тесту, можна переходити між запусками, та дивитися які були помилки до цього. Справа від опису Step знаходяться вкладення, де можна прикріплювати

картинки фрагментів коду тощо, які можна відкривати відразу у системі, просто натиснувши на відповідне вкладення, після цього відкриється вспливаюче вікно з контентом.

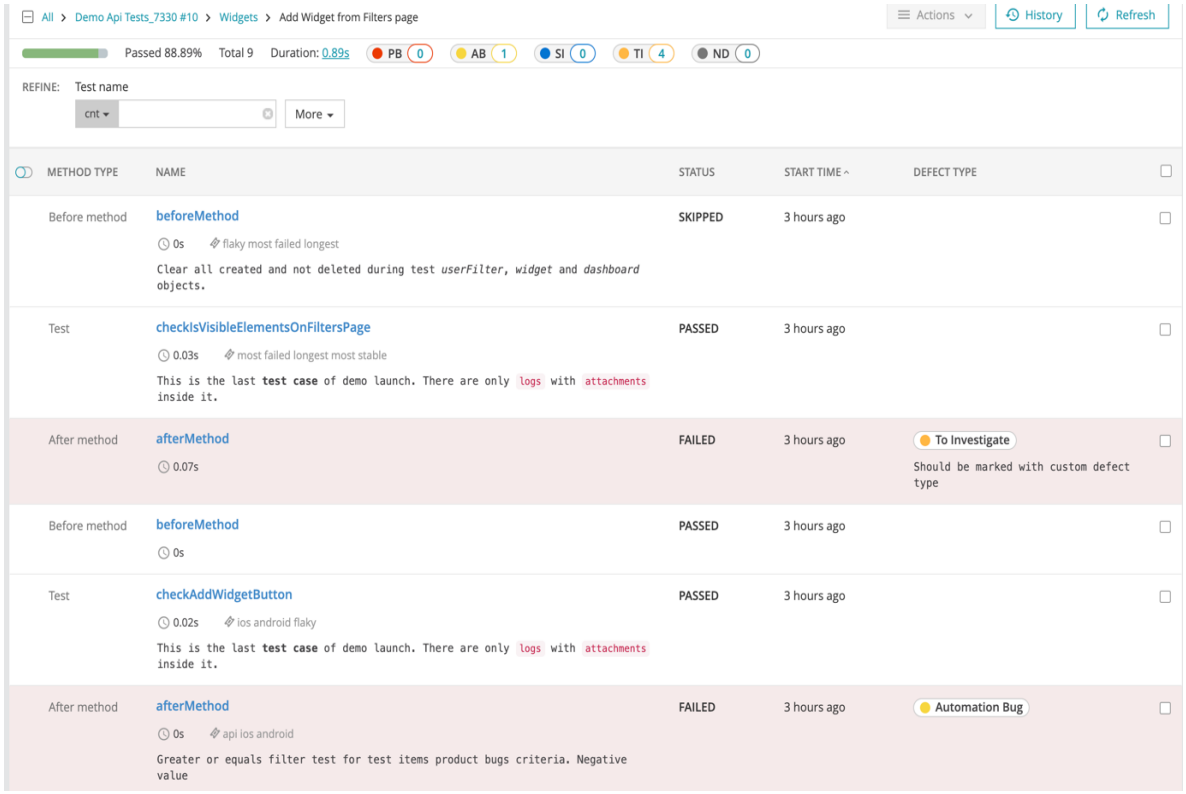


Рис. 3.9. Рівень test case

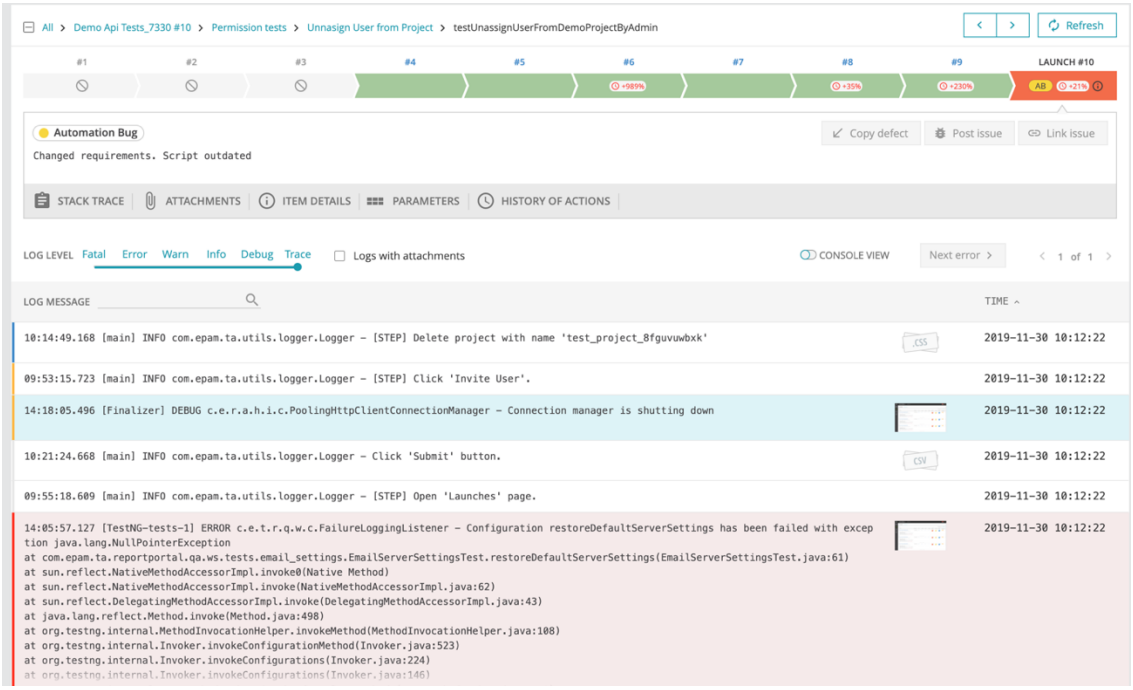


Рис. 3.10. Рівень test step

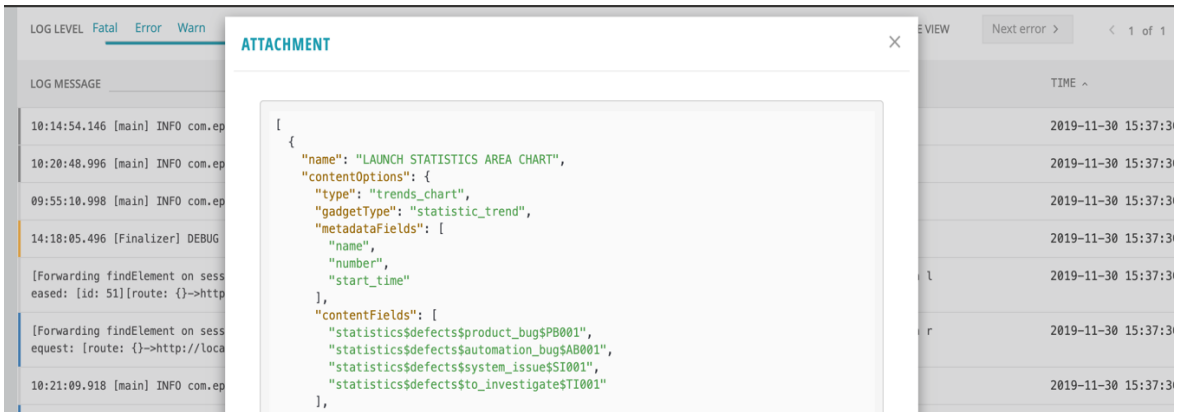


Рис. 3.11. Приклад відкриття вкладення

Якщо перейти до вкладки Stack trace, можна побачити весь stack trace, який і застосовувався у автоматичному аналізі.

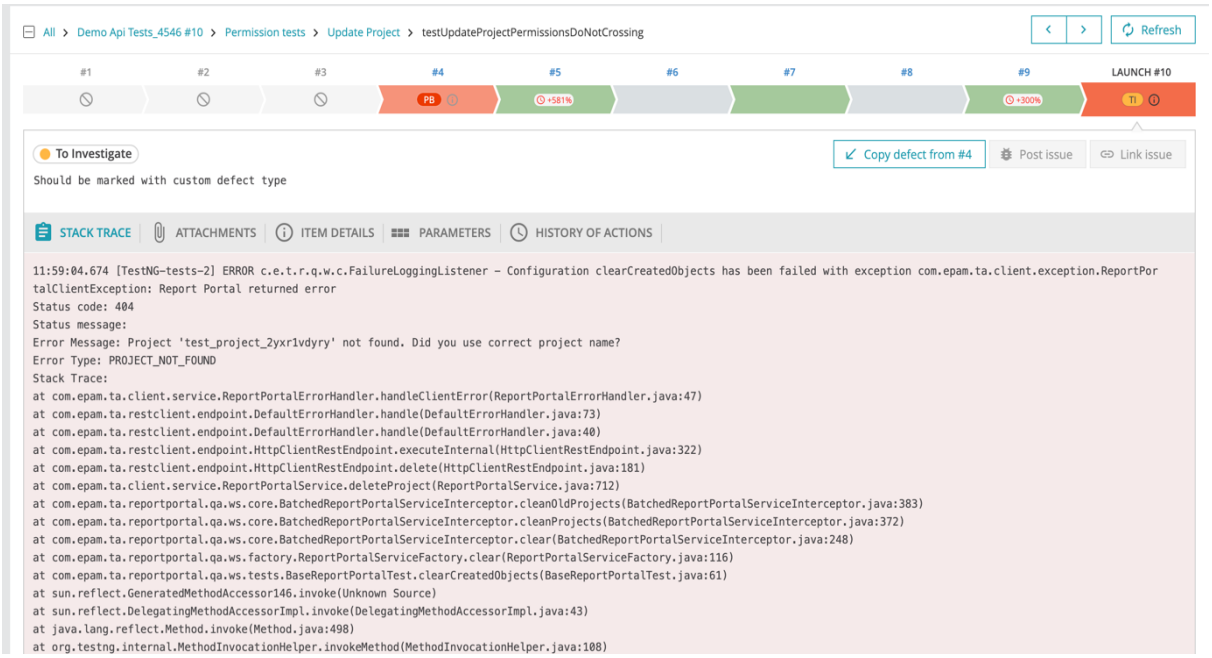


Рис. 3.12. Stack trace тесту

Система дозволяє поєднувати декілька запусків в один. Якщо проект має величезну кількість тестових наборів, вони розділяються на частини, так як не можуть бути в одному конкретному запуску. Як тільки вони будуть завершені, їх можна об'єднати в один окремий запуск, щоб представити ці дані на інформаційних панелях і створити звіти.

Реалізовано два типа злиття: лінійне (Linear) та глибоке (Deep). У випадку, якщо користувач вибере параметр "Лінійне злиття", створюється новий запуск. Новий запуск містить елементи злиття запусків. Рівні елементів залишаються такими ж, як у запуску початківців. Статистика стану та видань обчислюється як сума статистики всіх об'єднаних запусків. Початкові запуски видаляються з системи.

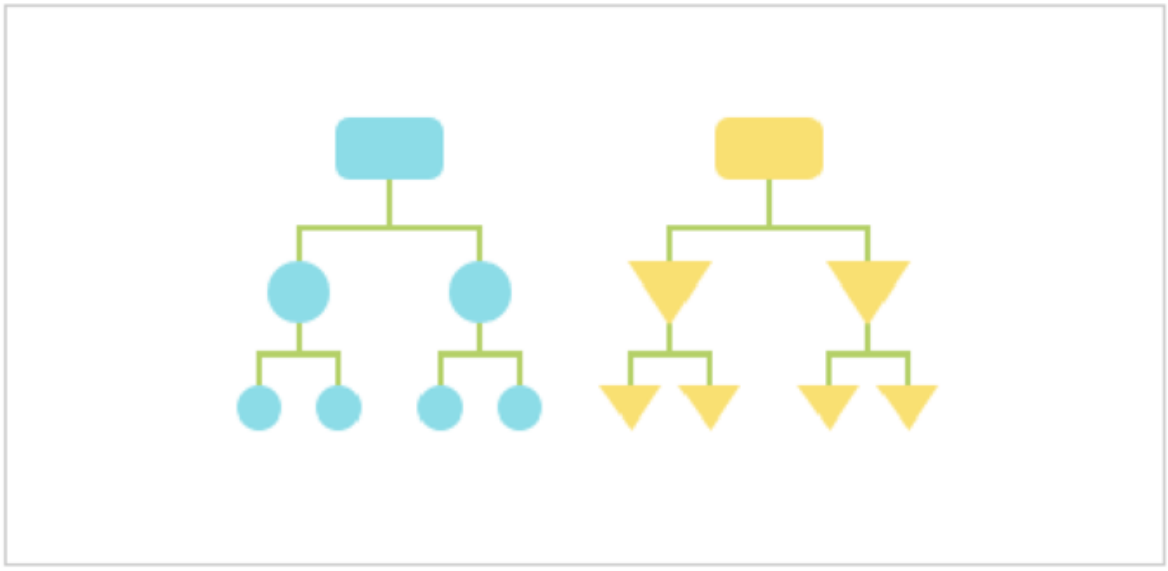


Рис. 3.13. Наочне зображення запусків перед злиттям

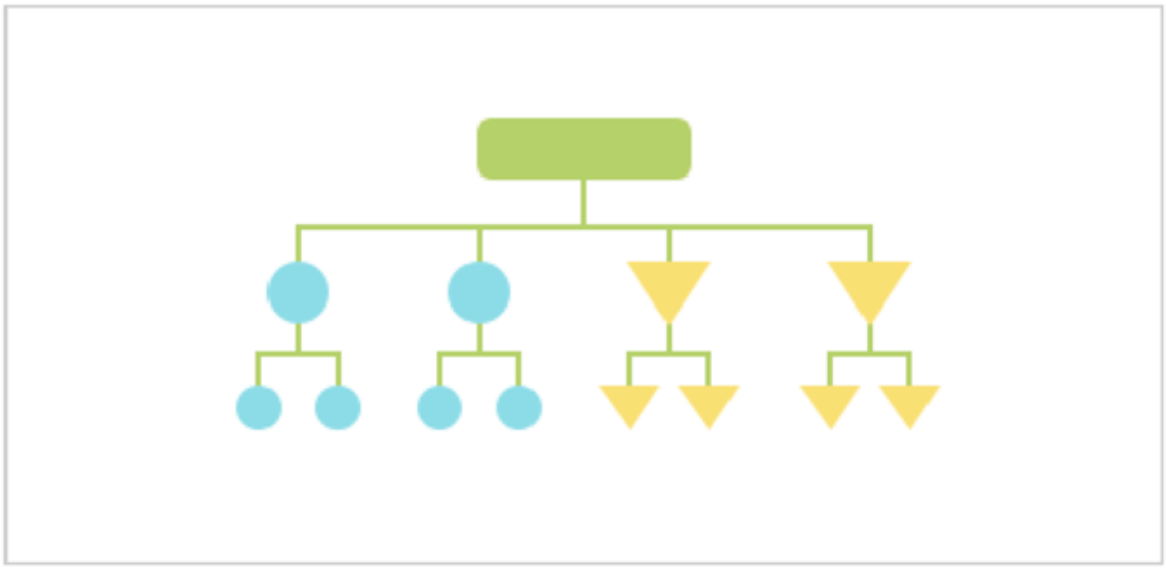


Рис. 3.14. Наочне зображення запуску після лінійного злиття

У випадку, якщо користувач вибере параметр "Глибоке злиття", система створює новий запуск та перевіряє елементи одночасно за наступними умовами:

- тестові елементи з однаковими назвами;
- тестові елементи, що мають один і той же тип (SUITE або TEST); тестові елементи знаходяться на одному шляху (кількість батьків);
- тестові елементи з нащадками. Якщо такі елементи знайдені, до нового запуску додається лише найдавніший;
- усі нащадки збираються на своїх рівнях, як і в оригінальних запусках. Злиття починається з верхніх рівнів до нижніх. Якщо верхній рівень не об'єднаний, нижній рівень також не буде об'єднаний. Елементи без нащадків не зливаються, незважаючи на рівень. Статистика стану та випусків обчислюється для нового запуску. Оригінальні запуски видаляються з системи.

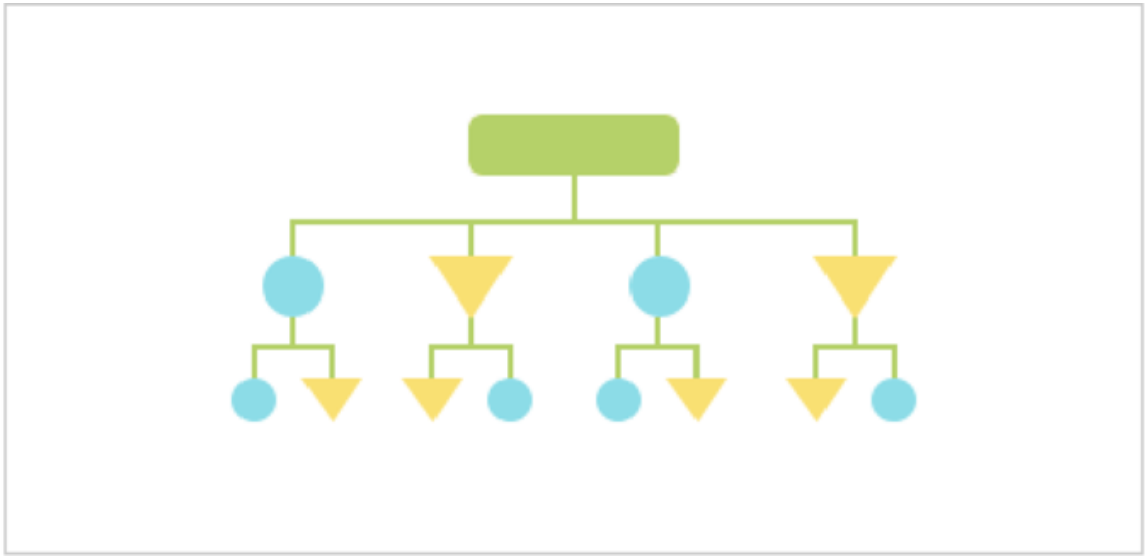


Рис. 3.15. Наочне зображення запусків після глибого злиття

Алгоритм лінійного та глибокого злиття, який описаний вище, показаний на рис. 3.16.

Наприклад, у нас є Launch-1 і Launch-2 для об'єднання. Якщо система знайде, що Suite_A у Launch-1 та Suite_A у Launch-2 мають однакові імена

та однакові типи та той самий шлях та мають нащадків, то до нового запуску додається лише найперший Suite_A (відповідно до часу початку). Усі нащадки об'єднаних номерів об'єднані під Suite_A. Потім система шукає таку саму відповідність на наступному рівні (рівень тесту).

Щоб об’єднати запуски, потрібно виконати наступні дії:

1. Перейти на сторінку "Запуски".
2. Вибрати необхідні запуски, натиснувши їхні прапорці.
3. Відкрити список "Дії".
4. Натиснути кнопку "Об'єднати".
5. Буде відкрито спливаюче вікно злиття.
6. Вибрати тип злиття "Лінійне злиття" або "Глибоке злиття".

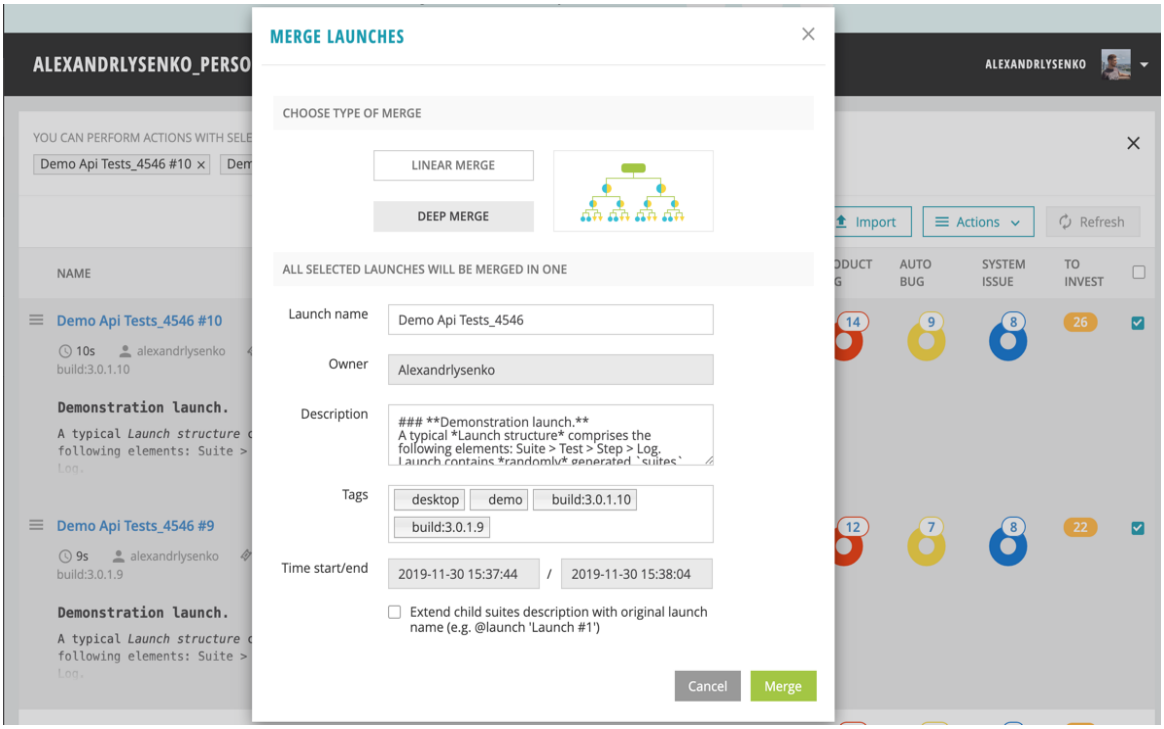


Рис. 3.17. Злиття запусків

Якщо елементи не відповідають умовам, описаним для параметра "Глибоке злиття", вони збираються так само, як описано для параметра "Лінійне злиття".

Система має гнучку систему фільтрування, користувач може фільтрувати результати виконання за типами помилок, запусками, за словами, що входять до певного запуску, можна створювати свої фільтри та зберігати їх для подальшого використання.

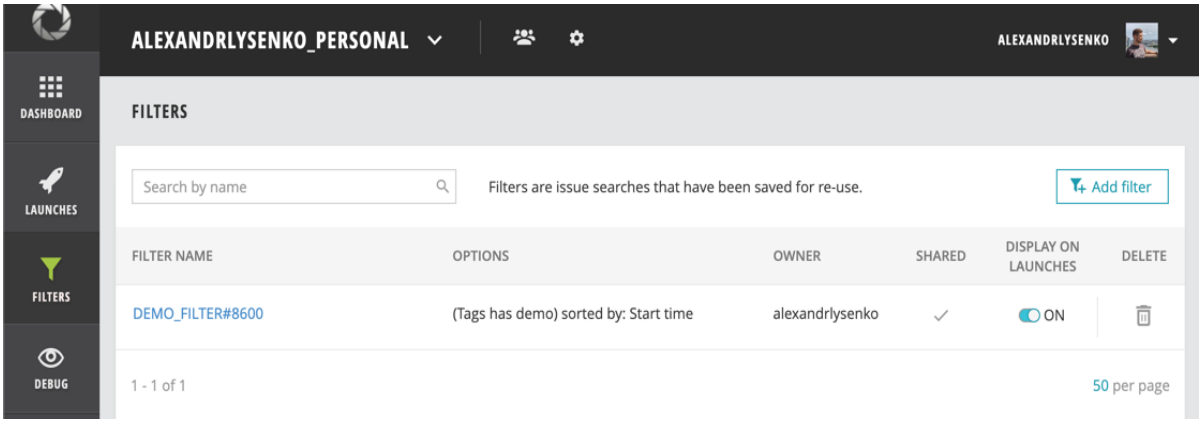


Рис. 3.18. Вкладка Фільтри

Для створення фільтру необхідно натиснути Add filter в правому кутку, після цього з’явиться меню створення фільтрів, як показано на рис. 3.18, де можна вибрати декілька варіантів фільтрації за наступними критеріями:

– назва – цей фільтр відображається завжди. Для цього пошуку потрібно щонайменше 3 символи. Система шукає запуски, які містять вказані символи у назві запуску. Наприклад, користувач встановлює Name = "abc", система виводить запуски з іменами "abcd", "zabc", "zabcd", "abc". Цей фільтр має наступні умови:
  - contains (фільтр покаже всі варіанти, в які входить даний рядок);
  - not contains (фільтр покаже всі варіанти, в які не входить даний рядок);
  - equals (фільтр покаже всі варіанти, що дорівнюють даному рядку;
  - not equals (фільтр покаже всі варіанти, що не дорівнюють даному рядку) ;

- номер – для цього пошуку потрібен мінімум один символ. Система здійснює пошук запусків із згаданим номером. Цей фільтр має наступні умови: дорівнює, більший або рівний; менше або рівне;
  - більший або рівний – показані всі тестові елементи з номером запуску, рівним або більшим за вказаний;
  - менше або рівне – показані всі тестові елементи з номером запуску, рівним або меншим за вказаний;
  - дорівнює – відображаються лише тестові предмети з вказаною кількістю;
- опис – для цього пошуку потрібно щонайменше 3 символи. Система шукає запуски, які містять вказані символи в описі запуску. Цей фільтр має наступні умови: contains, not contains, equals, not equals;
- час початку – система здійснює пошук запусків із заданим часом запуску. Доступні такі значення:
  - сьогодні – показані всі тестові завдання з часом початку Сьогодні; фільтр містить динамічне значення та пошукові запуски із початковим часом дорівнює сьогодні, тобто поточна дата;
  - останні 2 дні – показані всі тестові завдання з часом початку Останні 2 дні; фільтр містить динамічне значення, яке оновлюватиметься щодня, а пошукові запуски із початковим часом дорівнюють Останні 2 дні;
  - останні 7 днів – показані всі тестові завдання з часом початку Останні 7 днів; фільтр містить динамічне значення, яке оновлюватиметься щодня, а пошукові запуски із початковим часом дорівнюють Останні 7 днів;
  - останні 30 днів – показані всі тестові завдання з часом початку Останні 30 днів; фільтр містить динамічне значення, яке оновлюватиметься щодня, а пошукові запуски із початковим

часом становлять Останні 30 днів;

- спеціальний діапазон – відображаються всі тестові елементи із початковим часом у користувацькому періоді; користувач може встановити строгий період або динамічне значення (фільтр буде переміщуватися на один день щодня, де сьогодні (поточний день) – останній день у періоді);

– теги – можна шукати тег із принаймні одним символом у ньому. Система здійснює пошук запусків із зазначеними тегами. Можна вказати кілька тегів. Для цього необхідно натиснути на потрібний тег під назвою запуску, і система відобразить усі запуски, які мають цей тег. Цей фільтр має наступні умови:

- all – відображаються всі тестові елементи, які містять усі зазначені теги;
- without all –відображаються всі тестові елементи без вказаних тегів у будь-яких комбінаціях;
- any – відображаються всі тестові елементи, які містять будь-які комбінації зазначених тегів;
- without any – відображаються всі тестові елементи без усіх зазначених тегів у випадку, коли обидва теги присутні в тестовому елементі;
- усього, успішно завершені (passed), провалені (failed), пропущені (skipped), потребують визначення (to investigate). Мають наступні умови:
- більше або дорівнює – показані всі запуски із тестовими кейсами, рівними або більшими за вказані;
- менше або дорівнює – показані всі запуски з тестовими кейсами, рівними або меншими за вказані;
- дорівнює – показані лише запуски із заданою кількістю тестових випадків;
- product Bug, Automation Bug, System Issue. Перелік критеріїв

фільтрації залежить від використання типів дефектів проекту. Якщо використовується стандартний набір дефектів: помилка продукту, помилка автоматизації, системна помилка, можна побачити їх у списку відфільтрованих критеріїв. У проектах зі спеціальними дефектами система може фільтрувати запуски за кожним заздалегідь визначеним типом дефектів, кожним типом дефекту та загальною кількістю дефектів, що належать до однієї групи. Наприклад, якщо створюється 2 користувацькі помилки продукту, загальна кількість помилок продукту буде обчислюватися як сума попередньо визначеної помилки продукту + користувацька помилка продукту-1 та помилка продукту-2.

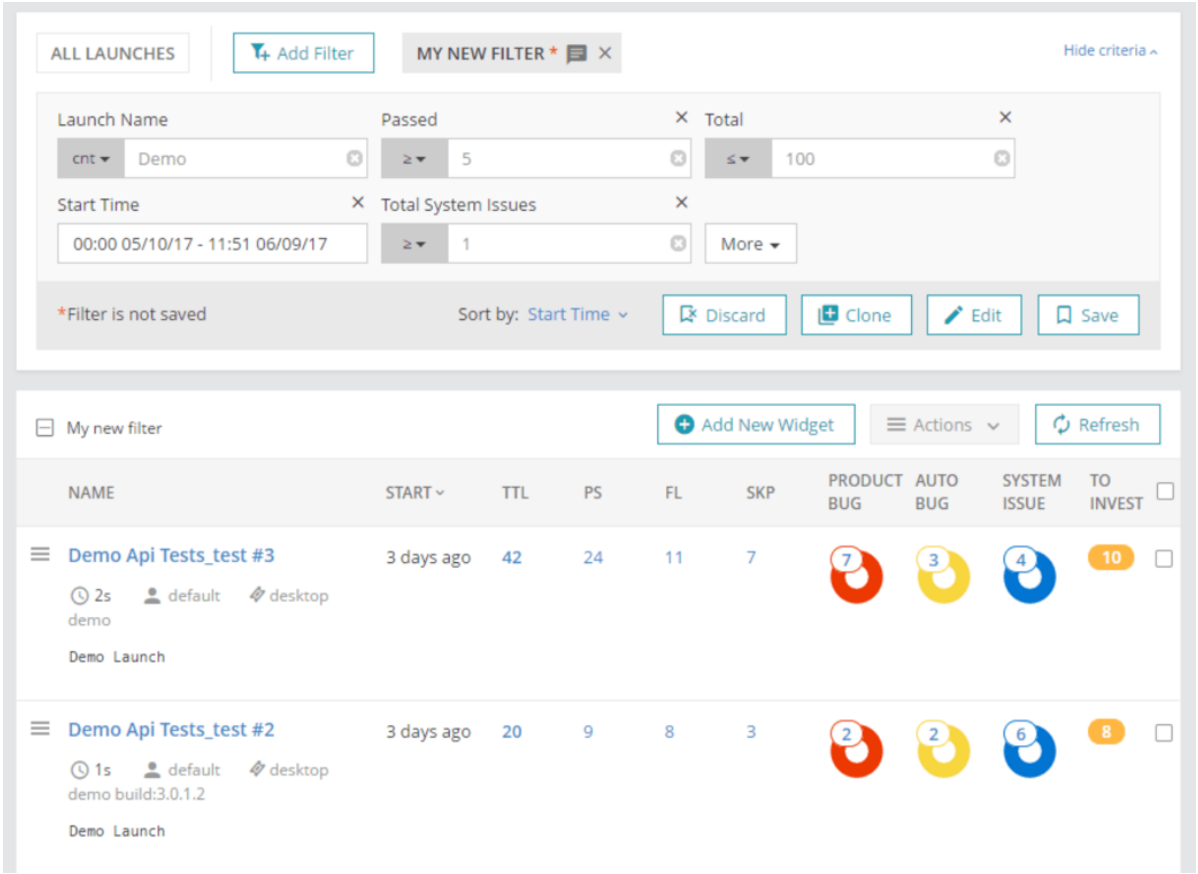


Рис. 3.19. Створення фільтрів

Після цього необхідно натиснути на Save, з’явиться модальне вікно, в якому можна дати назву фільтру та опис.

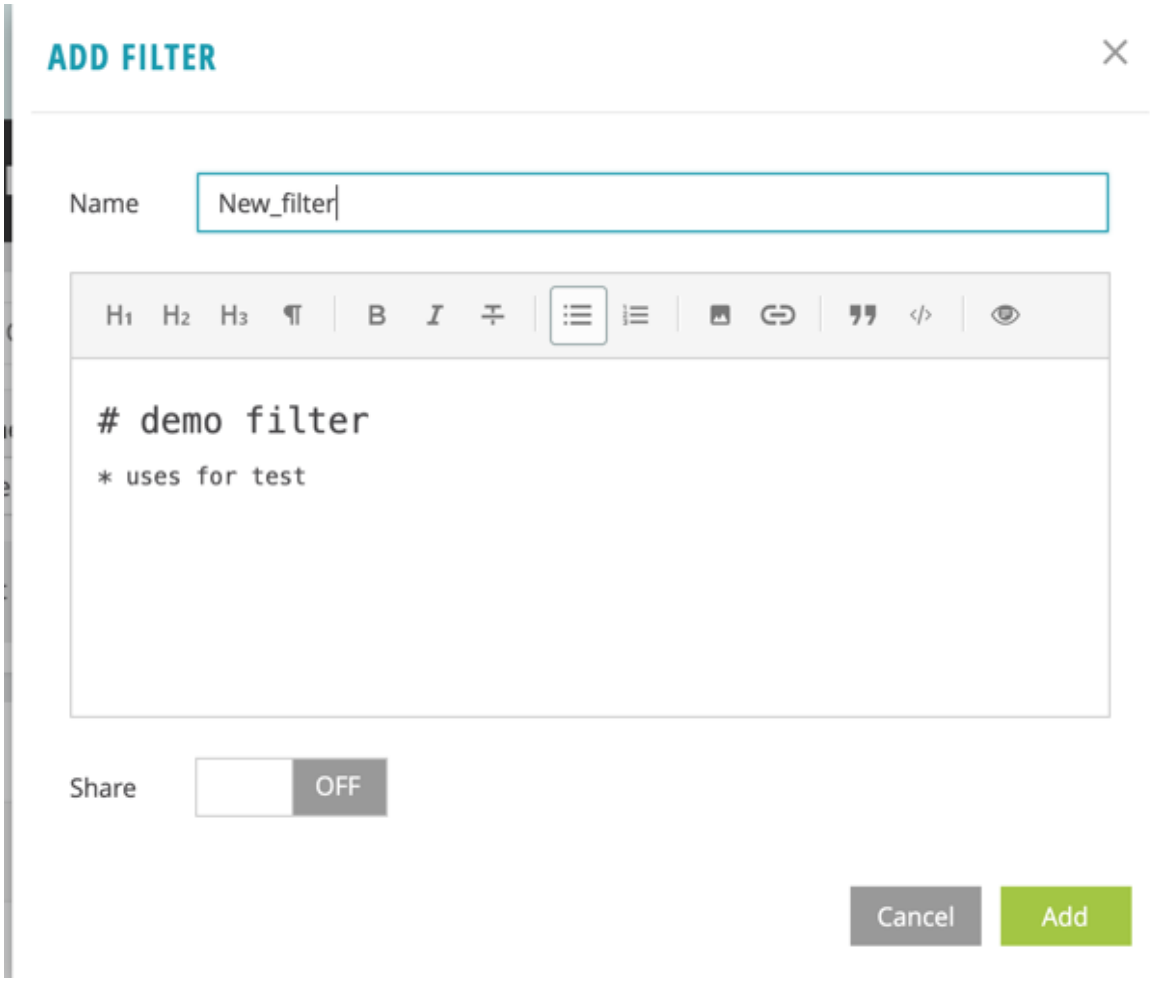


Рис. 3.20. Збереження фільтру

Також у системі передбачені налаштування для автоаналізу, що знаходяться у вкладці Project settings.

У цій секції користувач може виконати наступні дії:

1. Включити або виключити автоаналіз.
2. Обрати, для яких запусків застосовувати автоаналіз (для всіх / для запусків з однаковою назвою).
3. Налаштовувати параметри ElasticSearch.
4. Видаляти / створювати індекс ElasticSearch.

Також система надає можливість користувачам налаштувати 4 основні параметри ElasticSearch вручну:

– minDocFreq – мінімальна частота збережених логів у ElasticSearch

(індексі), у яких слід використовувати слово з аналізованого логу. Якщо кількість логів нижче вказаного значення, це слово буде ігноруватися для АА в аналізованому лозі. Чим частіше слово з'являється в індексі, тим менша його вага. Мінімальне значення 1, максимальне значення 10;

- minTermFreq – мінімальна частота слова в аналізованому лозі. Якщо кількість слів нижче вказаного значення, це слово ігнорується для АА. Чим частіше слово з'являється в аналізованому журналі, тим більше воно важить. Мінімальне значення 1, максимальне значення 10;
- minShouldMatch – відсоток рівності слів між аналізованим логом та конкретним логом ElasticSearch. Якщо для логу ElasticSearch значення менше, ніж встановлено, для АА цей лог буде проігноровано. Мінімальне значення 50, максимальне значення 100;
- кількість рядків журналу – кількість перших рядків стектрейсу, які слід врахувати в ElasticSearch. У програмі ElasticSearch буде збережено лише обрану кількість рядків стектрейсу. Можливі значення: 2, 3, 4, 5, Усі. Якщо ви вибрали "ВСЕ", повний текст стектрейсу буде збережено в індексі ElasticSearch.

Параметри MinDocFreq і MinTermFreque беруть участь у розрахунку коефіцієнта TF-IDF. Таким чином, можна нормалізувати лог і зменшити важливість частих, але марних слів у журналі (наприклад, "Java", прийменників, тощо) вручну.

Параметр MinShouldMatch бере участь у підрахунку балу. Це мінімальне значення для *coord*(*q*, *d*) (відсоток рівності слів між аналізованим логом та певним логом ElasticSearch). Таким чином, можна збільшити точність пошуку і вибрати необхідний мінімальний рівень подібності.

За допомогою кількості рядків журналу можна записати першопричину відмови тесту в перші рядки та налаштувати аналізатор для врахування лише необхідних рядків.

За допомогою цих 4 параметрів користувач може налаштувати точність аналізу, яка йому потрібна. Для зручності були підготовлені 3 попередні набори з наступними значеннями:

- light – умови пошуку не дуже "суворі". Ви отримаєте більше результатів, але з меншим рівнем подібності;
- moderate – "золота середина";
- classic – умови пошуку суворі. Ви отримаєте менше результатів, але з більш високим рівнем схожості.

Аналіз можна запустити вручну. Щоб розпочати аналіз вручну, необхідно виконати наступні дії:

1. Перейти на сторінку "Launches".
2. Виберати параметр "Аналіз" у контекстному меню поруч із вибраним іменем запуску.
3. Обрати область попередніх результатів, на основі яких тест-кейси необхідно автоматично проаналізувати. За замовчуванням вибрано на сторінці налаштування, але можна змінити його вручну.

За допомогою цього меню можна вибрати 3 варіанти:

1. Усі запуски.
2. Запуски з однаковою назвою.
3. Тільки поточний запуск.

Параметри "Усі запуски"та "Запуски з однаковою назвою" працюють так само, як і в налаштуваннях проекту. Якщо вибрати тільки поточний запуск, система аналізує тестові елементи обраного запуску лише на основі вже дослідженої дати цього запуску.

У випадку, якщо користувач обрав позицію "Для дослідження" елементів, система аналізує лише елементи з дефектом типу "Для дослідження" у вибраному запуску.

У випадку, якщо користувач вибирає елементи, що аналізуються автоматично (AA), система аналізує лише ті елементи, які вже були проаналізовані за допомогою автоматичного аналізу. Результати попереднього запуску аналізу будуть видалені, а елементи будуть проаналізовані ще раз.

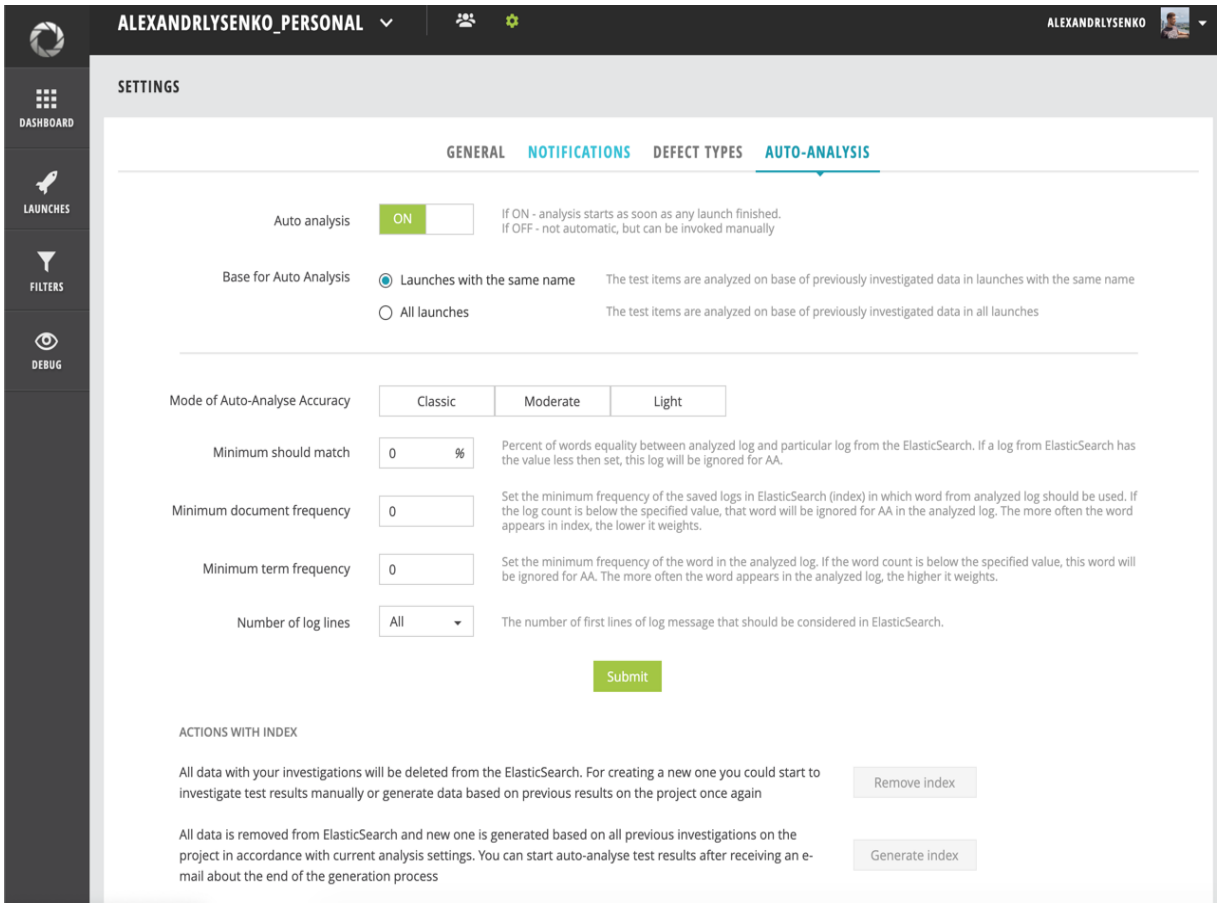


Рис. 3.21. Налаштування автоаналізу

Якщо користувач вибирає елементи, що аналізуються вручну – система аналізує лише ті елементи, які користувач вже проаналізував вручну. Результати попереднього запуску аналізу будуть видалені, а елементи будуть проаналізовані ще раз.

У разі їх комбінації – система аналізує результати в залежності від обраних варіантів.

Будь-які запуски з активним процесом аналізу будуть позначені міткою "Аналіз".

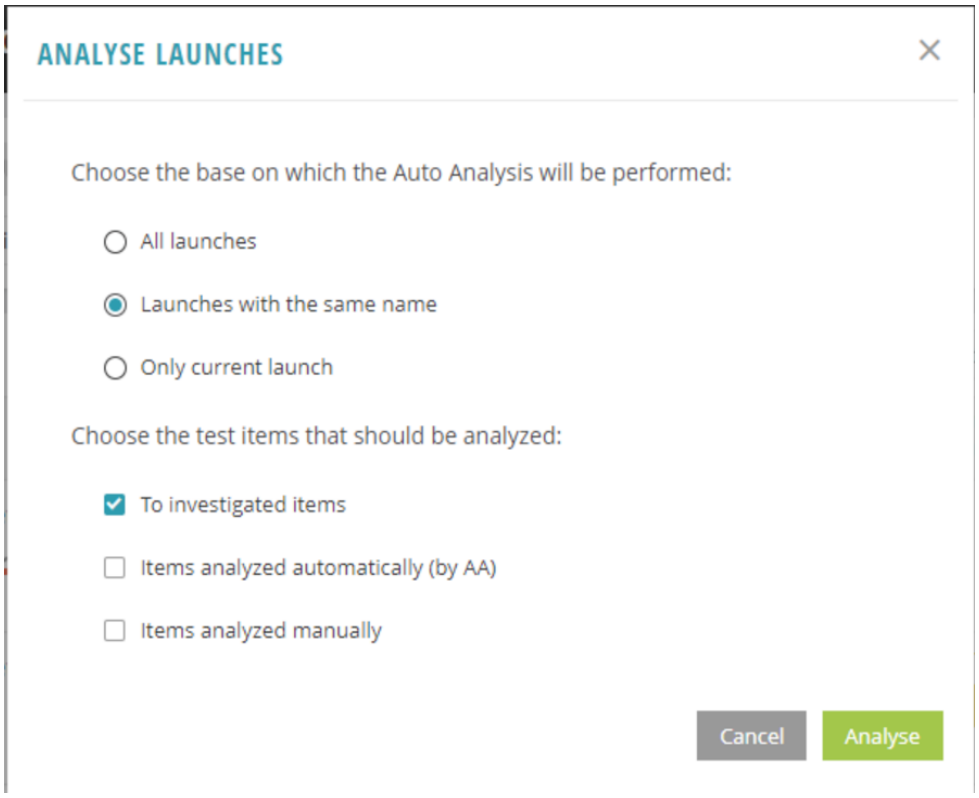


Рис. 3.22. Ручний запуск автоаналізу

Коли тестовий елемент аналізується системою, на тестовому елементі на рівні кроку встановлюється мітка "AA". Ви можете фільтрувати результати за параметром "Аналізується (AA)".

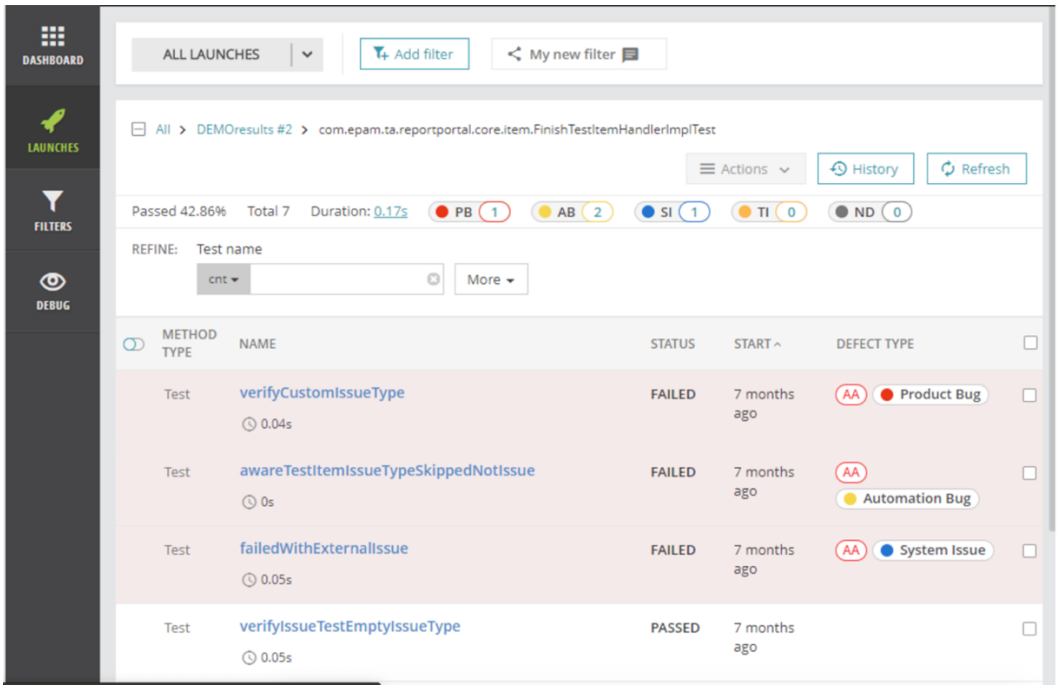


Рис. 3.23. Мітка автоматичного аналізу

Система також дозволяє створювати віджети. Віджети містять спеціальні графічні елементи управління, які були розроблені, щоб забезпечити простий у використанні спосіб відображення та аналізу результатів процесу автоматизації тестування.

Віджети можна додати до інформаційних панелей на вкладці "Інформаційні панелі". Віджети будуть видимі в рамках проекту, в якому вони створені.

Щоб створити новий віджет, необхідно виконати наступні дії:

1. Перейти на сторінку "Усі інформаційні панелі" та створити нову інформаційну панель або обрати існуючу.
2. Натиснути кнопку "Додати новий віджет".
3. Відкриється майстер віджетів. Для того, щоб додати новий віджет, необхідно виконати наступне:
   a. Обрати шаблон віджета.
   b. Обрати фільтр зі списку нижче або створити новий фільтр. Функціональність для пошуку допомагає швидше знайти фільтр. Обрати інші параметри віджетів: Критерії, елементи, режим запуску або часової шкали (якщо можна застосовано до вибраного шаблону віджету).
4. Після того, як виконані всі дії, натиснути кнопку "Зберегти". Новий віджет буде доданий на панель віджетів у верхній частині.

   Віджети автоматично оновлюються щохвилини.

Наразі у системі є 15 шаблонів віджетів:

- Launch statistics chart;
- Overall statistics panel;
- Launches duration chart;
- Launch execution and issue statistic;
- Project activity panel;
- Test-Cases growth trend chart;
- Investigated percentage of launches;

– Launches table;
– Unique bugs table;
– Most failed test cases table;
– Failed cases trend chart;
– Non-Passed test-cases trend chart;
– Different launches comparison chart;
– Flaky test cases table (TOP-20).

Launch statistics chart – може використовуватися в двох режимах: режимі запуску та режимі часової шкали:

1. Віджет у режимі запуску показує тенденцію зростання кількості тестових випадків із кожним вибраним статусом від запуску до запуску.
2. Віджет у режимі часової шкали відображає суму тестових випадків із кожним вибраним статусом, розподіленим за датами.

Параметри віджета.

1. Фільтр: потрібно принаймні один фільтр.
2. Кількість елементів: 1-150. Значення за замовчуванням – 50.
3. Критерії віджетів: Усі критерії вибрані за замовчуванням.
4. Режим: запуск або часова шкала.
5. Вид: площа / стовпчики.
6. Критерії віджетів: Усі критерії вибрані за замовчуванням.

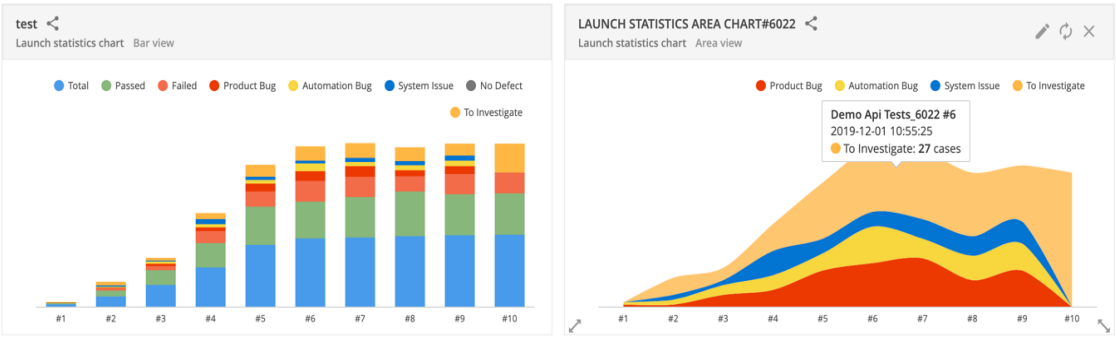


Рис. 3.24. Launch statistics chart в режимі запуску

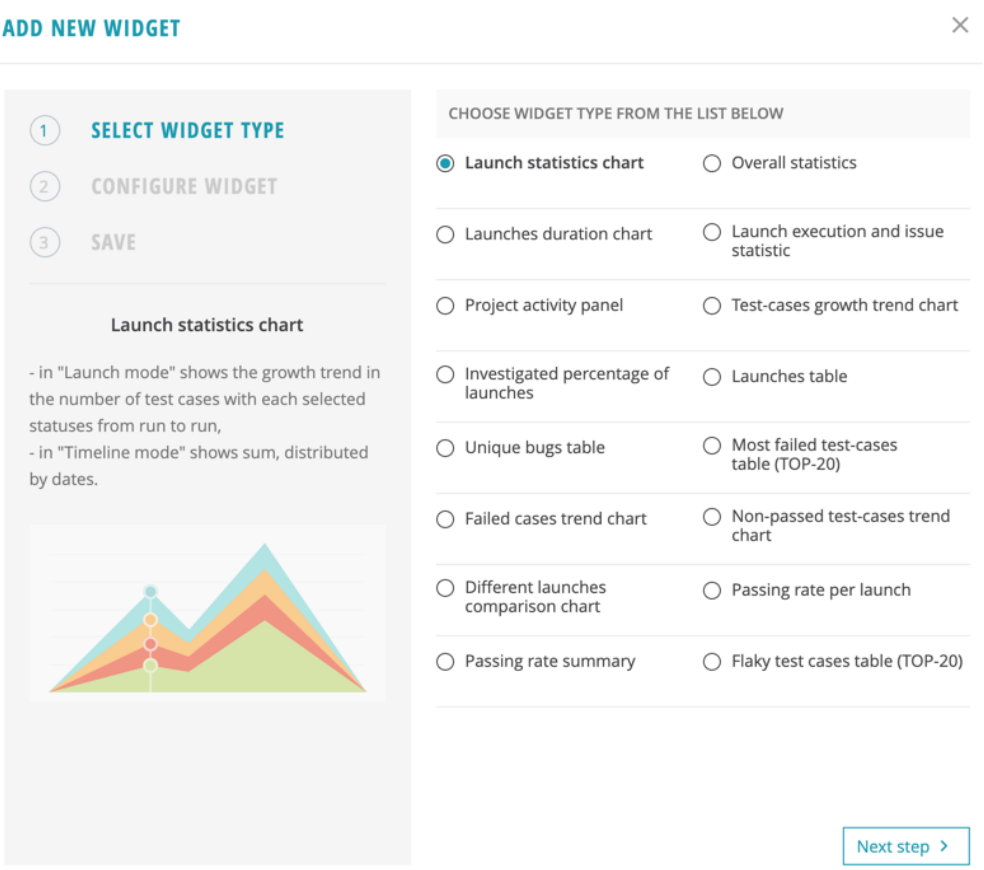


Рис. 3.25. Панель створення віджетів

Віджет містить інформацію із вибраними статусами; можна натиснути на стан, щоб видалити / додати його до діаграми.

У режимі запуску:

– вісь X відображаються номери запуску та відображаються назві запусків при наведенні курсора;
– вісь Y показує суму тест кейсів з кожним вибраним статусом.

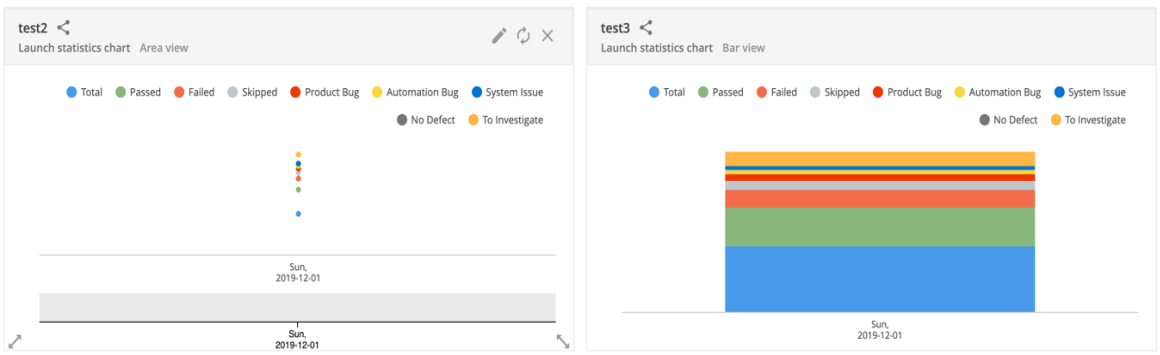


Рис. 3.26. Launch statistics chart в режимі часової шкали

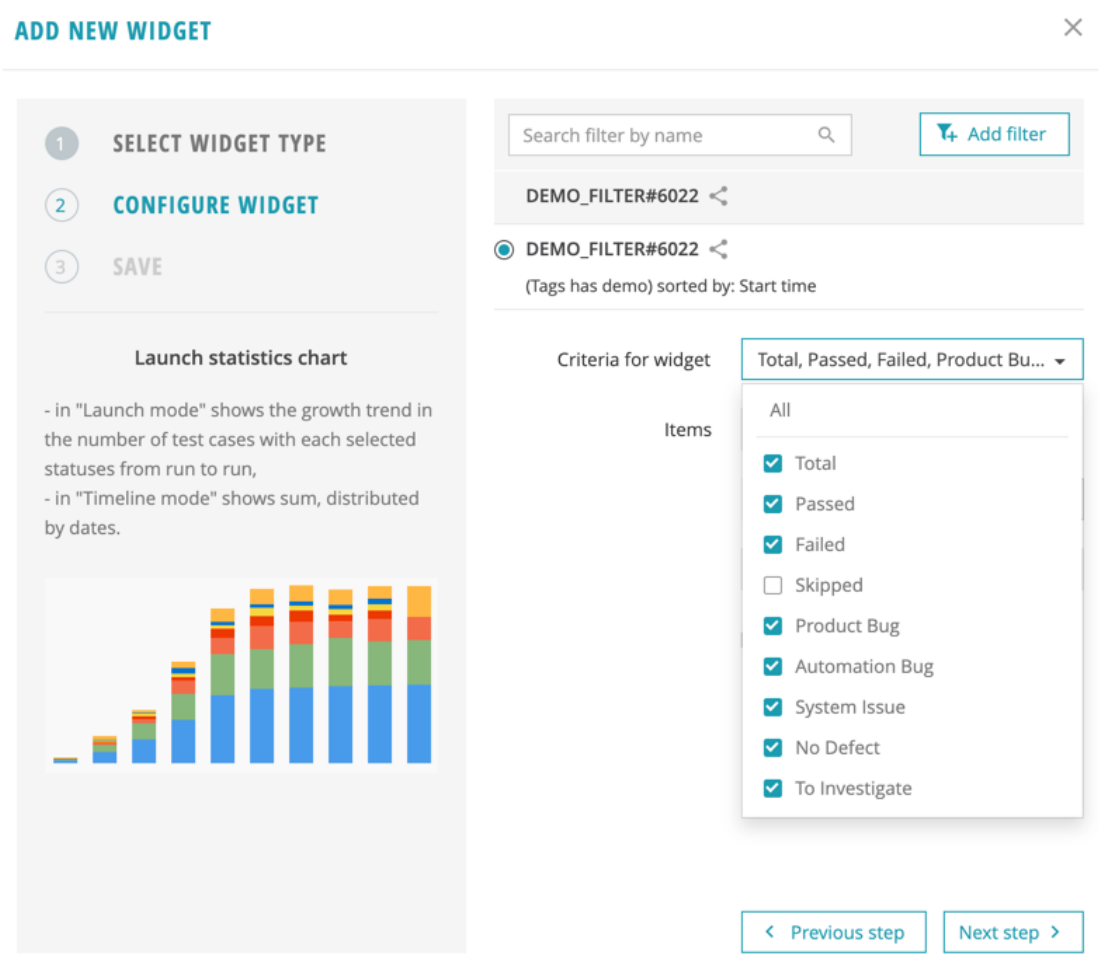


Рис. 3.27. Створення віджету Launch statistics chart

У режимі часової шкали:

– на осі X відображаються дати та дні тижня;

– вісь Y показує суму статистики запусків з кожним вибраним статусом, розподілену по днях.

Overall statistics panel – на панелі відображається сума тест-кейсів із кожним статусом для кожного вибраного запуску.

Параметри віджета:

1. Фільтр: потрібно принаймні один фільтр.
2. Кількість елементів: 1-150. Значення за замовчуванням – 50.
3. Критерії віджетів: Усі критерії вибрані за замовчуванням.
4. Режим: запуск або часова шкала.
5. Вид: панель / кругова діаграма.
6. Критерії віджетів: Усі критерії вибрані за замовчуванням.

Віджет показує статистику всіх запусків / або останніх запусків вибраного фільтра. Статистика поділяється на наступні розділи:

– Skipped, Passed, Failed;
– Product Bug, System Issue, Automation Bug, No Defect and To Investigate.

Статистика кожного типу відображається у відсотках. Після наведення курсора відображається точне число для кожного типу. У віджеті є розділи, на які можна натискати, після натискання на вказаний розділ у віджеті, система спрямовує вас на відображення для відповідного вибору.

Якщо обрати режим усіх запусків, віджет покаже статистику всіх запусків у фільтрі. Щоб переглянути лише останні виконання кожного запуску, слід вибрати останні запуски.

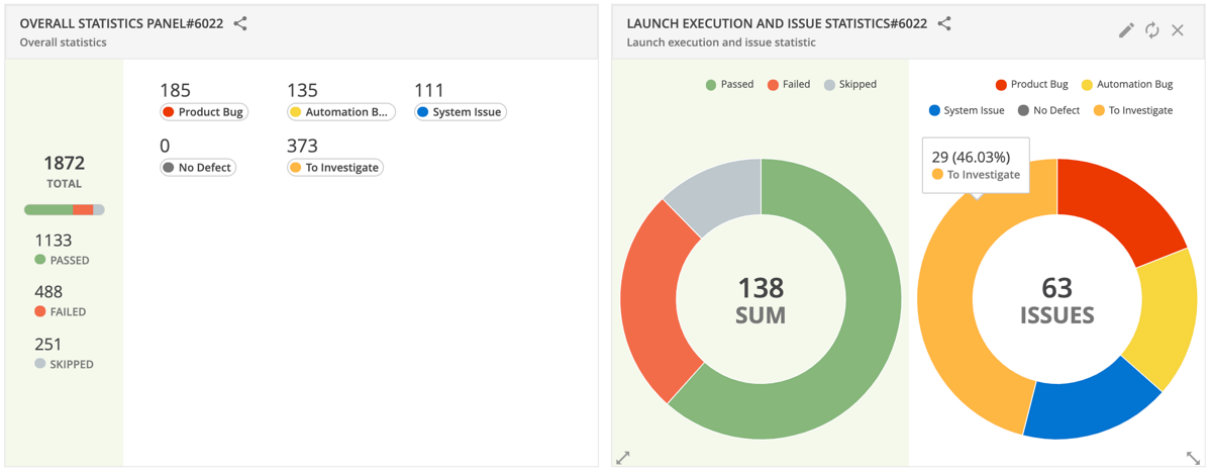


Рис. 3.28. Overall statistics panel

3.5. Висновки до розділу 3

У даному розділі описаний процес розроблення програмного забезпечення для аналізу результатів тестування за допомогою машинного навчання, а саме kNN алгоритму, який дозволяє скоротити час аналізу помилок та переглядати стан виконання тестів у режимі реального часу.

Описані вимоги користувача, функціональні вимоги та можливості розробленої системи.

Наведено архітектуру, на основі якої виконувалось розроблення програмного забезпечення, інтерфейс користувача. Проаналізовано сучасні мови програмування, середовища розробки та бібліотеки.

Для реалізації запропонованого програмного методу було обрано стек технологій.

Для частини backend використано мову програмування Java, у якості бази даних була використана MongoDB.

Для частини frontend були використані технології HTML, CSS, JS, фреймворк Angular та Node.Js.

4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Порівняння отриманих результатів тестування програмного забезпечення різними методами

У розробленій системі були порівняні 3 методи кластеризації а саме:

– метод Байєса;

– метод опорних векторів;

– метод *k*-найближчих сусідів.

Головною характеристикою порівняння цих методів була точність, так як основна ціль розробленої системи – зменшити часові витрати та людські ресурси на аналіз результатів тестування програмного забезпечення. Результати порівняння цих методів можна побачити у таблиці 4.

Таблиця 4

Порівняння точності методів кластеризації

Кількість провалених тестів	Метод *k*-найближчих сусідів	Метод опорних векторів	Метод Байєса
100	0.7483	0.7948	0.8126
200	0.8347	0.8589	0.7650
500	0.9078	0.8451	0.7607
1000	0.9820	0.9241	0.7520

Як бачимо з таблиці, метод *k*-найближчих сусідів трохи "програє" у точності методу опорних векторів на маленьких наборах даних в середньому на 3-6%, проте на наборах 500-1000 метод *k*-найближчих сусідів показує результати на 6-10% кращі результати, ніж метод опорних векторів та на 23-26% кращі результати ніж метод Байєса.

Проведено порівняння с ручним аналізом "провалених" тестів та з використанням методів машинного навчання на реальному проекті. Було запущено аналіз 150-ти тестів, алгоритм впорався з цим завданням приблизно за 20 хвилин, та не зміг автоматично розпізнати всього 9 тестів, в той час як людині знадобилося приблизно 4 години, щоб перечитати всі помилки та віднести їх до категорій. Таким чином, алгоритм впорався з цією задачею у 12 разів швидше, ніж людина.

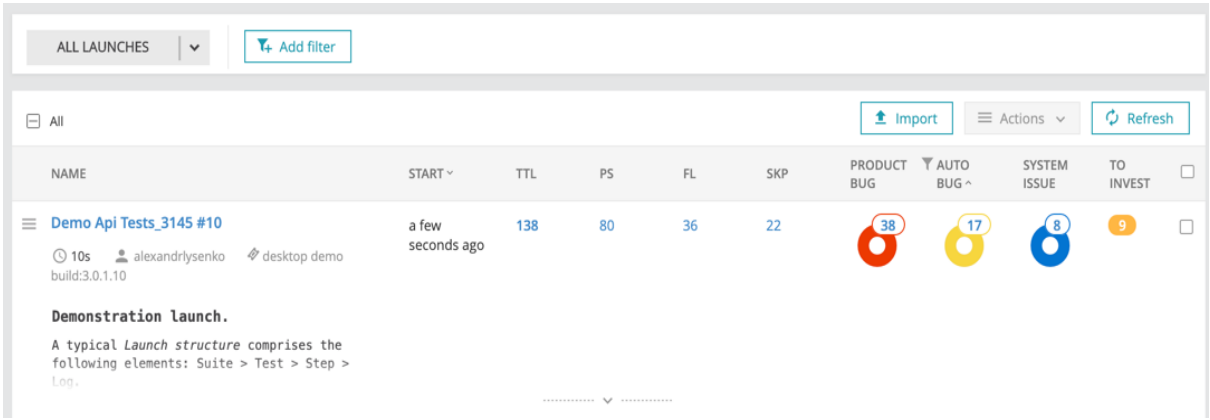


Рис. 4.1. Результат аналізу з використанням алгоритму

4.2. Подальше вдосконалення розробленого програмного забезпечення

У майбутньому планується додати підтримку для різних мов програмування. Над даний момент система працює з лише з мовою програмуваня Java та фреймворком для автоматизації тестування JUnit.

Також планується додати інтеграцію з баг-трекінговими системами таких як:

- Jira;
- Trello;
- Bugzilla;
- Airbrake;
- ZenHub.

4.3. Висновоки до розділу 4

У даному розділі наведено результати порівняння методів кластеризації *k*-найближчих сусідів, опорних векторів та Байєса для автоматичного аналізу результатів тестування програмного забезпечення на 100, 200, 500 та 1000 провалених тестів. Також було проведено порівняння автоматичного аналізу результатів з ручним.

При дослідженні результатів порівняння автоматичного аналізу помилок програмного забезпечення при 1000 провалених тестах метод *k*-найближчих сусідів показує точність 0,982, що є найбільш точним результатом серед розглянутих методів кластеризації. Також при аналізі 150 провалених тестів алгоритм показав результат у 12 разів швидший, аніж ручний аналіз.

Таким чином показано, що запропонований програмний метод є ефективним.

5. ПОБУДОВА БІЗНЕС-МОДЕЛІ

5.1. Виділення проблеми

На сьогодні тестування займає один з ключових етапів розроблення програмного забезпечення. Для аналізу розроблюваного програмного забезпечення потрібні першокласні тестувальники, їх постійне навчання та мотивація, що для невеликих і середніх компаній є значною проблемою. Розробники програмного забезпечення намагаються уникати ручного тестування продукту з метою зниження ризику впливу людського фактору та зниження вартості тестування. Середній час аналізу помилок після автоматичного тестування для 500 "провалених" тестів може досягати до 24 год., тобто 4 робочі дні.

При розробленні продукту необхідно, щоб після його випуску він працював коректно, без збоїв та відповідав вимогам замовника. Бажано, щоб уже у процесі розроблення частково функціонуючий продукт працював згідно усіх вимог щодо функціональності та був масштабованим. Це завдання частоково допомагає вирішити тестування. ІТ-компанії потребують автоматизації процесу тестування, подальшого аналізу та виправлення помилок у ПЗ для надання замовнику повної картини того, що відбувається у проекті з детальним описом того, що вже працює та потребує доопрацювання. Це дуже складно робити вручну, особливо тоді, коли проект дуже великий і у ньому виконується більше тисячі тестів. В цьому допомогає автоматизація тестування, тобто написання автоматичних тестів, які можна буде виконувати повторно. Таким чином, після додавання нової функціональності в додаток можна бути впевненим, що все працює без збоїв, хоча і тут не все відбувається так гладко як хотілося.

Виникають проблеми з менеджерської сторони відносно аналізу результатів тестування програмного зебезпечення. Покриття тестами зростає надто повільно і ні замовник, ні команда цього не бачить, а потім через півроку з’ясовується, що замовник сподівався на вдвічі більше покриття. Деколи відбувається так, що тести були реалізовані але не

запущені, гроші витрачено на цю роботу, але цінність нульова. Наступний момент, коли тести були написані, запущені але не проаналізовані, також цінність нульова, так як гроші було витрачено, але результат ніхто не перевірив. Також написані тести можуть бути нестабільними і команда витрачає час на виправлення цих тестів та повторний їх запуск, витрачається в два рази більше грошей, що також не дуже добре для замовника та бізнесу.

Для повноти розуміння побудуємо дерево проблем, на якому будуть зображені проблеми, з якими може зустрітися користувач. Дерево проблем наведено на рис. 5.1.

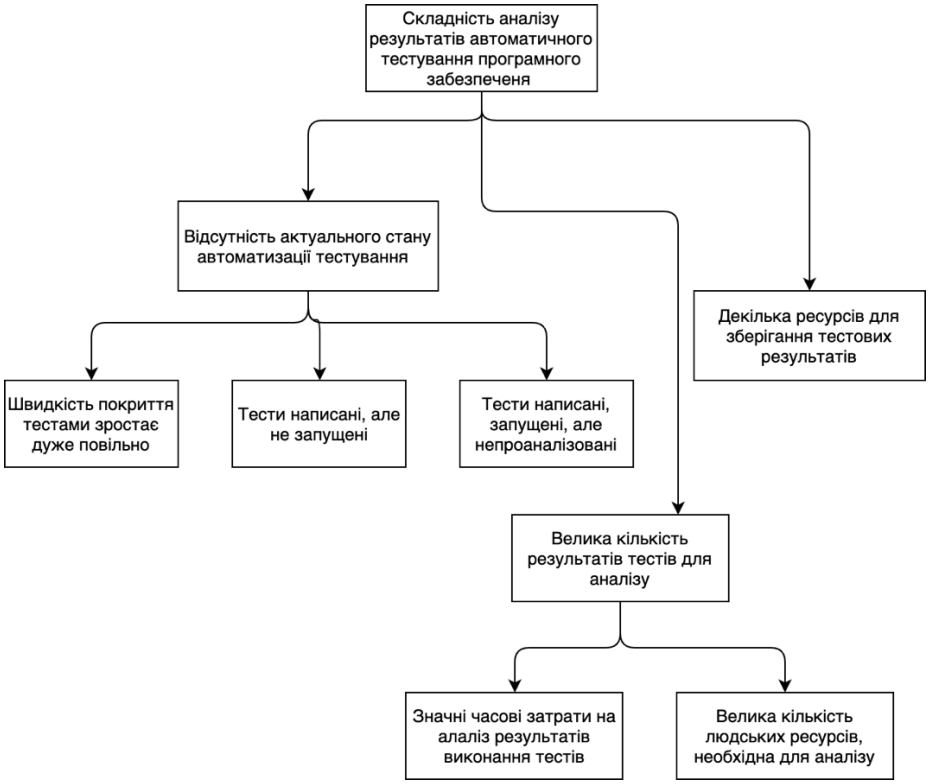


Рис. 5.1. Дерево проблем

У середніх та великих компаніях існує така проблема, як дуже велика кількість тестів для аналізу (більше тисячі), на аналіз яких може витрачатися багато часу (більше 6 робочих днів), що дуже уповільнює робочий процес. Відповідно, необхідно багато людських ресурсів для аналізу цих помилок. Ще одною проблемою є зберігання результатів тестування для подальшої обробки та надання команді задач на віправлення цих помилок.

Саме тому виникає потреба у розробленні програмної системи для автоматизованого аналізу результатів тестування програмного забезпечення, яка дозволить скоротити час аналізу помилок та переглядати стан виконання тестів у режимі реального часу.

5.2. Зацікавлені сторони

В процесі створення бізнес-моделі проекту було сформовано групи зацікавлених в реалізації проекту сторін:

- власники програмного забезпечення;
- проектні менеджери;
- розробники (спільнота та компанії з розробки);
- компанії, що надають послуги на тестування програмного забезпечення.

Основною зацікавленою стороною є власники програмного забезпечення. Для них дуже важливо витратити якнайменше коштів на розроблення програмного забезпечення, при цьому продукт має повністю функціонувати та бути покритим тестами. Тому для них необхідно бачити прогрес розроблення програмного забезпечення, щоб планувати наступні кроки, такі як реклама та подальше вдосконалення продукту.

Також категорією зацікавлених сторін є проектні менеджери, оскільки їм необхідно координувати та налагоджувати роботу команди. Їм вкрай необхідно грамотно планувати роботу команди для реалізації нової функціональності, виправлення критичних помилок, коригувати дії команди, яка дуже важлива для клієнта в конкретний момент часу, слідкувати за "здоров’ям" системи для вчасної реакції та виправлення помилок, а також планування показу нової функціональності кінцевому клієнту.

Ще однією зацікавленою стороною є власне розробники програмного забезпечення. Безпосередньо вони виконують розроблення програмного забезпечення. Для них також важливо бачити, які помилки відбуваються в

їх коді, вчасно виправляти їх, давати менеджерам оцінку на виконання тієї чи іншої задачі.

Наступною категорією зацікавлених сторін є компанії, що надають послуги на тестування програмного забезпечення. Для них також важливо правильно планувати час на виконання тестування, розподіляти людські ресурси на аналіз результатів тестування.

У табл. 5 зведено усі групи зацікавлених сторін, їх інтереси, та вплив (міра зацікавленості у вирішенні наявних проблем).

5.3. Комерційне рішення. Основні характеристики

Відповідно до вище зазначених проблем, можна описати кінцевий продукт, що має їх вирішувати. Даний програмний продукт буде реалізовано, як описано у попередніх розділах, у вигляді веб-додатку, який буде використовувати метод *k*-найближчих сусідів для автоматичного аналізу результатів тестів програмного забезпечення. Цей додаток можна розгорнути на будь-якому сервері завдяки Docker.

Таблиця 5

Зацікавлені сторони

Зацікавлена сторона	Інтерес зацікавленої сторони	Вплив зацікавленої сторони	Стратегії приваблення зацікавлених сторін
Власники програмного забезпечення	Зменшення кількості людських ресурсіх на аналіз результатів тестування програмного забезпечення, зменшення часу на аналіз тестування,	Високий	Виступи на конференціях та IT-форумах Надання можливості користування технологією

Продовження табл. 5

Власники програмного забезпечення	Перегляд актуального стану автоматизації на проекті		Виступи на конференціях та IT-форумах Надання можливості користування технологією безкоштовно за визначеними умовами
Проектні менеджери	Правильний розподіл команди на проекті, грамотне планування часу для реалізації нової функціональності, перегляд актуального стану автоматизації на проекті	Високий	Виступи на конференціях та IT-форумах Надання можливості користування технологією безкоштовно за визначеними умовами
Розробники користувацьких додатків	Вчасне реагування на виникнення помилок у тестах та швидке їх виправлення	Низький	Виступи на конференціях та IT-форумах
Компанії, що надають послуги з тестування програмного забезпечення	Зменшення людських ресурсів, необхідних для аналізу результатів тестування програмного забезпечення	Високий	Виступи на конференціях та IT-форумах

Завдяки цьому можна дуже просто інтегрувати автоматизацію проекту до системи, що зменшить час на підготовку.

Зміни будуть помітними у більшій мірі для власників програмного забезпечення, так як зменшиться час та людські ресурси на вирішення проблеми, визначеної у підрозділі 5.1. Саме тому клієнтом даного продукту є власники програмного забезпечення, а співпраця буде побудована на моделі співробітництва бізнесу для бізнесу, або як він ще називається B2B.

5.4. Конкурентні переваги рішення

Як вже зазначалося вище, бізнес зацікавлений у зменшенні вартості розробки програмного забезпечення та зменшенні кількості людських ресурсів необхідних для його розроблення. Тому зацікавлені у використанні в розробленні програмного забезпечення таких систем, що дозволять переглядати стан розробки програмного продукту, покриття тестами, зберігати результати виконання тестів в одному централізованому сховищі.

Наразі існують такі системи, що допомагають організувати автоматизацію тестування програмного забезпечення. Прикладом можуть бути такі системи як Selenium, Katalon Studio, Allure report. Але дані системи не є кросплатформенними, підходять далеко не для всіх типів програмного забезпечення, досить складно налаштовуються та мають ряд недоліків.

Отже, конкурентними перевагами програмного продукту, що пропонується, є:

- єдине сховище для результатів виконання тестів;
- інтеграція та розгортання системи на виділеному сервері;
- можливість автоматичного аналізу результатів тестування програмного забезпечення;
- створення користувацьких типів та підтипів помилок для кластеризації помилок;
- історія запусків тестів;
- кросплатформність;

– перегляд виконання автотестів у режимі реального часу;
– створення фільтрів для відображення тільки потрібних результатів;
– зручна статистика автоматизації у вигляді таблиць та графіків.

5.5. Унікальна ціннісна пропозиція

Ціннісна пропозиція – це пояснення того, як продукт вирішує проблему. Його можна скласти за формулою:

Ціннісна пропозиція = Проблема + Рішення / Продукт.

У дереві проблем було виділено низку проблем, а у зацікавлених сторонах – визначено очікування відповідних сторін від продукту. Власники програмних продуктів бажать отримати готовий програмний продукт якомога швидше, при цьому продукт має повністю функціонувати без збоїв, бути покритим автоматичними тестами. Також важливо, щоб інвестиції були окуплені цим продуктом, тому важливо зменшувати час на тестування за рахунок допоміжних систем, що, в свою чергу, знизить кількість людських ресурсів та час, необхідний для аналізу результатів тестування і подальшого виправлення помилок. Менеджери проектів – грамотне розподілення людськими ресурсами на проекті, відслідковування стану покриття тестами на проекті, швидке реагування на критичні помилки та негайне їх виправлення. Розробники – бачити, які помилки відбуваються в їх коді, вчасно виправляти їх, давати менеджерам оцінку на виконання тієї чи іншої задачі. Компанії, що надають послуги з тестування програмного забезпечення – правильно планувати час на виконання тестування, розподіляти людські ресурси на аналіз результатів тестування.

Дійсно, запропоноване рішення дозволяє частково задовольнити всі з наведених вище вимог зацікавлених сторін та вирішити наведені вище проблеми.

Отже, основною унікальною ціннісною пропозицією є суттєве зменшення часу на аналіз результатів тестування програмного забезпечення, зменшення людських ресурсів необхідних для аналізу.

Основними перевагами запропонованого рішення є простота розгортання системи на окремому сервері, можливість переглядати стан автоматизації у режимі реального часу.

5.6. Доходи та витрати

Дохід складається з продажу ліцензії на використання програмного забезпечення.

Продаж програмного забезпечення планується шляхом продажу ліцензії на програмне забезпечення. Слід зазначити, що ліцензія буде в декількох варіантах (повна та з відсутністю автоматичного аналізу).Також тарифи будуть відрізнятися в залежності від терміну використання та призначення цільового власницького програмного продукту: комерційний чи не комерційний.

Оцінюючи витрати на проект, варто виділити дві основні категорії: стартові і щомісячні витрати. Стартові витрати – це гроші, які вкладаються один раз на початку проекту, а щомісячні, відповідно, вкладаються в розвиток проекту щомісяця.

До стартових витрат відноситься:

- реклама (маркетинг);
- на юридичні послуги;
- на ріелтерські послуги, мінімальні меблі, канцелярію.

До щомісячних витрат відносяться:

- зарплати за посадами;
- оренда офісного приміщення;
- комунікації;
- податки;
- непередбачені витрати.

Детальніше ознайомитися з витратами на реалізацію проекту та прогнозованим прибутками можна з табл. 6.

Таблиця 6

Витрати на реалізацію проекту

Найменування витрат	1-й місяць, т. $	2-й місяць, т. $	3-й місяць, т. $	4-й місяць, т. $	5-й місяць, т. $	6-й місяць, т. $
Загальні витрати	7.5	1.5	1.5	1.3	1.7	1.2
Стартові витрати	7	1	1	0.5	0.5	
Реклама (маркетинг)	1	1	1	0.5	0.5	
Обладнання офісу	5					
Щомісячні витрати	0.5	0.5	0.5	0.8	1.2	1.2
ЗП				0.2	0.5	0.5
Оренда приміщення та комунікації	0.5	0.5	0.5	0.6	0.7	0.7
Заплановані прибутки			1	2	2	5
Результат (без оподаткування)	-7.5	-1.5	-0.5	0.7	0.3	3.8
Найменування витрат	7-й місяць, т. $	8-й місяць, т. $	9-й місяць, т. $	10-й місяць, т. $	11-й місяць, т. $	12-й місяць, т. $
Загальні витрати	1.2	1.2	1.2	1.2	1.2	1.2
Стартові витрати	1.2	1.2	1.2	1.2	1.2	1.2
Реклама (маркетинг)	0.5	0.5	0.5	0.5	0.5	0.5
Юридичні послуги	0.7	0.7	0.7	0.7	0.7	0.7

5.7. Бізнес-модель

Узагальнимо, все написане вище у лаконічну бізнес-модель у вигляді lean canvas.

Споживачі: власники програмного забезпечення, проектні менеджери.

Проблема: складність аналізу результатів автоматизації тестування програмного забезпечення: відсутність актуального стану автоматизації тестування; декілька ресурсів для зберігання тестових результатів; швидкість покриття тестами зростає дуже повільно; тести написані, але не запущені; тести написані, запущені, але непроаналізовані; велика кількість результатів тестів для аналізу; значні часові затрати на аналіз результатів виконання тестів; велика кількість людських ресурсів, необхідна для аналізу.

Рішення: розроблення веб-застосунку для автоматичного аналізу результатів тестування програмного забезпечення.

Унікальна ціннісна пропозиція: система, що за допомогою методу *k*-найближчих сусідів дозволяє автоматично аналізувати результати тестів та відносити помилки до категорій, що визначені користувачем.

Потоки доходів: доходи від продажу ліцензій; доходи від підтримки програмного забезпечення.

Структура витрат: утримання персоналу для надання технічної підтримки (виплати заробітних плат, соціальних виплат); утримання робочих місць для персоналу (оплата за оренду офісу та комунальні послуги); податкові витрати; оплата послуг юриста, бухгалтера; витрати на маркетинг.

Також в канву бізнес-моделі включаються структурні блоки: прихована перевага (перевага, яку неможливо скопіювати або купити), ключові метрики (основні показники, що вимірюються) та канали (шляхи до користувачів).

Канали: через платформи замовлення проектів; конференції з автоматизації тестування програмного забезпечення.

Ключові метрики: кількість проданих ліцензій.

Прихована перевага: простота підключення до існуючого проекту;

Таблиця 7

Канва бізнес-моделі

<table>
<tr>
<td rowspan="2">Проблема

складність аналізу результатів автоматизації тестування програмного забезпечення

відсутність актуального стану автоматизації тестування;

значні часові затрати на алаліз результатів виконання тестів;

велика кількість людських ресурсів, необхідна для аналізу;</td>
<td>Рішення

розроблення веб-застосунку для автоматичного аналізу результатів тестування програмного забезпечення</td>
<td rowspan="2">Унікальна ціннісна пропозиція

система, що за допомогою методу k-найближчих сусідів дозволяє автоматично аналізувати результати тестів та відносити помилки до категорій, що визначені користувачем</td>
<td>Прихована перевага

простота підключення до існуючого проекту;

швидке розгортання на виділеному сервері</td>
<td rowspan="2">Споживачі

власники програмного забезпеченя, що хочуть зменшити час та кількість ресурсів на тестування та аналіз результатів тестування

компанії, що надають послуги з тестування програмного забезпечення</td>
</tr>
<tr>
<td>Ключові метрики

кількість проданих ліцензій</td>
<td>Канали

через платформи замовлення робіт з ПЗ;
конференції;
бізнес-тренінги</td>
</tr>
</table>

Продовження табл. 7

Структура витрат	**Потоки доходів**
утримання персоналу; утримання робочих; податкові витрати; витрати на послуги юриста, бухгалтера; витрати на маркетинг.	доходи від продажу ліцензій; доходи від підтримки програмного забезпечення.

Отже, зважаючи на дані у табл. 7, можна зробити висновок, що запропонована система, що використовує метод *k*-найближчих сусідів для автоматичного аналізу результатів виконання тестів програмного забезпечення, а також відображення стану автоматизації проекту в реальному часі, має перспективи у своїй подальшій реалізації. Звичайно, проведений аналіз не враховує всіх ризиків та факторів, таких, як специфіка оподаткування у країні ведення бізнесу, проте навіть наявних досліджень достатньо, щоб прогнозувати комерційний успіх продукту та його окупність.

5.8. Висновки до розділу 5

У даному розділі проведено аналіз поточної ситуації у сфері автоматизації тестування, виявлено наявні проблеми та узагальнено їх у відповідному дереві проблем. Наряду з проблемами було виділено основні зацікавлені сторони у вирішенні існуючих недоліків, ступінь впливу даних сторін на вирішення проблем. Як наслідок, було запропоновано комерційне рішення з конкурентними перевагами, що задовольняє інтереси зацікавлених осіб та виділено унікальну ціннісну пропозицію запропонованого продукту. Проведено аналіз майбутніх клієнтів, досліджено сегменти ринку споживання. У результаті була описана бізнес-модель, що обґрунтовує доцільність реалізації даного продукту та прогнозує його потенційну окупність та прибутковість в подальшому.

ВИСНОВКИ

Автоматизація тестування програмного забезпечення є досить актуальною задачею на сьогодні. Як відомо, в процесі розроблення програмних продуктів програми піддаються змінам. Як би добре вони не були написані спочатку, в них вносяться зміни, зумовлені необхідністю виправлення існуючих помилок, виявлених в процесі їх виконання, або ж бажанням внести в програму додаткові зміни. Розширення галузей застосування існуючих програм призводить до появи нових функціональних вимог, які не були враховані спочатку. В цьому процесі виникає необхідність у написанні тестів для перевірки роботи тієї чи іншої функціональності. Функціональність накопичується дуже швидко, тому збільшується кількість тестів та помилок, які необхідно виправляти. Великі та середні компанії намагаються уникати ручного тестування, так як для цього необхідна велика кількість людських ресурсів. В результаті власники програмного забезпечення переходять на автоматизацію тестування програмного забезпечення.

У даній магістерській дисертації було поставлено за мету створити програмний метод для автоматичного аналізу результатів тестування програмного забезпечення на основі машинного навчання, а саме з використанням kNN-алгоритму, що значно зменшить час та людські ресурси, необхідні для аналізу результатів тестування. Результатом дослідження веб-додаток, який використовує даний алгоритм для автоматичної кластеризаціїї результатів тестування програмного забезпечення.

У першому розділі проаналізовано існуючі методи кластеризації для дослідження результатів тестування програмного забезпечення, виявлено їх переваги та недоліки. На основі досліджень та результатів аналізу було сформовано вимоги до програмного методу.

Основними вимогами стали: точність кластеризації; можливість створення категорій та підкатегорій для кластеризації помилок; можливість переглядати стан автоматизації тестування у реальному часі.

У другому розділі магістерської дисертації було детально описано роботу алгоритму відповідно до поставленої задачі. Основна ідея запропонованого підходу полягає у тому, що використовується траса стеку у якості даних, які необхідно кластеризувати, адже саме ця частина має у собі інформацію про причину "падіння" тесту.

У третьому розділі описані вимоги користувача, функціональні вимоги та можливості розробленої системи; наведено архітектуру, в основі якої виконувалось розроблення програмного забезпечення, інтерфейс користувача; проаналізовано сучасні мови програмування, середовища розробки та бібліотеки. Обрано стек технологій для реалізації методу.

У четвертому розділі описано результати порівняння методів кластеризації *k*-найближчих сусідів, опорних векторів та методу Байєса для автоматичного аналізу результатів тестування програмного забезпечення на 100, 200, 500 та 1000 "провалених" тестів. При дослідженні результатів порівняння автоматичного аналізу помилок програмного забезпечення при 1000 "провалених" тестах метод *k*-найближчих сусідів показує точність 0,982, що є найкращим результатом серед розглянутих методів кластеризації. Також при аналізі 150 "провалених" тестів алгоритм виконав аналіз результатів тестування у 12 разів швидше, ніж ручний аналіз. Таким чином показано, що запропонований програмний метод є ефективним.

У п'ятому розділі створено бізнес-модель кінцевого продукту, виділено основні зацікавлені сторони у вирішенні існуючих недоліків, ступінь впливу даних сторін на вирішення проблем. Запропоновано комерційне рішення з конкурентними перевагами, що задовольняє інтереси зацікавлених осіб та унікальну ціннісну пропозицію запропонованого продукту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Angular – What is Angular? [Електронний ресурс]. – Режим доступу: https://angular.io/docs. Дата доступу: травень 2018. Назва з екрану.
2. The 5 Clustering Algorithms Data. Режим доступу: https://towardsdata science. com/the-5-clustering-algorithms-data-scientists
3. A. K. Jain. Algorithms for Clustering Data. / A. K. Jain., R. C. Dubes – Prentice Hall, IGI Global, 2012 – pp. 43-62.
4. Евклідова відстань [Електронний ресурс]. – Режим доступу: https://stud.com.ua/75021/statistika/osnovni_zahodi_vidstaney.
5. Відстань Чебишева [Електронний ресурс]. – Режим доступу: https://helpiks.org/6-9937.html
6. L. Kaufman. Finding Groups in Data: An Introduction to Cluster Analysis. / L. Kaufman, P. J. Rousseeuw. – MA: MIT Press, 2007 – pp. 215–266.
7. kNN algorithm. Режим доступу: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
8. Java stack trace, understanding and using for debug: Режим доступу: https://www.scalyr.com/blog/java-stack-trace-understanding/
9. Inverse Document Frequency. Режим доступу: https://nlp.stanford.edu/IR-book/html/htmledition/inverse-document-frequency-1.html.
10. Elasticsearch.Режим доступу: https://www.elastic.co/guide/en/ elasticsearch /reference/current/indices-analyze.html.
11. K Means Clustering with Tf-idf Weights. Режим доступу: http://jonathanzong.com/blog/2013/02/02/k-means-clustering-with-tfidf-weights.

12. Дюран Б. Кластерный анализ. / Дюран Б., Оделл П. – К.: Статистика, 1999. – 128-84с.
13. React – A JavaScript library for building user interfaces [Електронний ресурс]. – Режим доступу: https://reactjs.org/.
14. Vue.js [Електронний ресурс]. – Режим доступу: https://vuejs.org/. Дата доступу: травень 2018. Назва з екрану.
15. REST – Glossary | MDN [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Glossary/REST. Дата доступу: травень 2018. Назва з екрану.
16. Python [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Python. Дата доступу: травень 2018. Назва з екрану.
17. Python переваги та недоліки [Електронний ресурс]. – Режим доступу: http://www.codingclub.net/Articles/Python/Primenenie_yuzika_Python_prei muschestva_i_nedostatki. Дата доступу: травень 2018. Назва з екрану.
18. Ruby [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Ruby. Дата доступу: травень 2018. Назва з екрану.
19. Ruby переваги та недоліки [Електронний ресурс]. – Режим доступу: https://www.slideshare.net/guest5f907e/top10-ruby. Дата доступу: травень 2018. Назва з екрану.
20. PHP [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/PHP. Дата доступу: травень 2018. Назва з екрану.
21. PHP переваги та недоліки [Електронний ресурс]. – Режим доступу: http://www.programmersforum.ru/showthread.php?t=164538. Дата доступу: травень 2018. Назва з екрану.

22. Node.js [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Node.js. Дата доступу: травень 2018. Назва з екрану.
23. Cassandra [Електронний ресурс]. – Режим доступу: http://cassandra.apache.org/. Дата доступу: травень 2018. Назва з екрану.
24. РСУБД [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/RSUBD. Дата доступу: травень 2018. Назва з екрану.
25. Cassandra переваги та недоліки [Електронний ресурс]. – Режим доступу: https://habr.com/post/155115/. Дата доступу: травень 2018. Назва з екрану.
26. MongoDB [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/blog/post/thinking-documents-part-1. Дата доступу: травень 2018. Назва з екрану.
27. MongoDB vs Cassandra [Електронний ресурс]. – Режим доступу: https://blog.panoply.io/cassandra-vs-mongodb. Дата доступу: травень 2018. Назва з екрану.
28. MongoDB переваги та недоліки[Електронний ресурс]. – Режим доступу: http://qaru.site/questions/92527/pros-and-cons-of-mongodb. Дата доступу: травень 2018. Назва з екрану.
29. CouchDB [Електронний ресурс]. – Режим доступу: http://couchdb.apache.org/. Дата доступу: травень 2018. Назва з екрану.
30. CouchDB переваги та недоліки [Електронний ресурс]. – Режим доступу: https://habr.com/post/123338/. Дата доступу: травень 2018. Назва з екрану.
31. Мандель И. Д. Кластерный анализ. / Мандель И. Д. – К.: Финансы и статистика, 1988. – 10 с.

32. Жамбю М. Иерархический кластер-анализ и соответствия. / Жамбю М. – К.: Финансы и статистика, 1988. – 345 с.
33. Дюран Б. Кластерный анализ. / Дюран Б., Оделл П. – К.: Статистика, 1999. – 128-84с.
34. Gorban A.N. Principal Graphs and Manifolds, Ch. 2 in: Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques. / Gorban A.N., Zinovyev A.Y. – MA: MIT Press, 2010 – pp. 125– 140.
35. P. S. Bradley. "Clustering via Concave Minimization, " in Advances in Neural Information Processing Systems, vol. 9. / P. S. Bradley., O. L. Mangasarian, W. N. Street. MA: MIT Press, 1997– pp. 368–374.
36. A. K. Jain. Algorithms for Clustering Data. / A. K. Jain., R. C. Dubes – Prentice Hall, IGI Global, 2012 – pp. 43-62.
37. L. Kaufman. Finding Groups in Data: An Introduction to Cluster Analysis. / L. Kaufman, P. J. Rousseeuw. – MA: MIT Press, 2007 – pp. 215–266.

Додаток 1

Лістинг програми

Лістинг 1. baseModules.js

```
var requirejs, require, define;
(function(ba) {
    function G(e) {
        return "[object Function]" === K.call(e)
    }

    function H(e) {
        return "[object Array]" === K.call(e)
    }

    function v(e, t) {
        if (e) {
            var n;
            for (n = 0; n < e.length && (!e[n] || !t(e[n], n, e)); n +=
1);
        }
    }

    function T(e, t) {
        if (e) {
            var n;
            for (n = e.length - 1; - 1 < n && (!e[n] || !t(e[n], n, e));
n -= 1);
        }
    }

    function t(e, t) {
        return fa.call(e, t)
    }

    function m(e, n) {
        return t(e, n) && e[n]
    }

    function B(e, n) {
        for (var r in e)
            if (t(e, r) && n(e[r], r)) break
    }

    function U(e, n, r, i) {
        return n && B(n, function(n, s) {
            if (r || !t(e, s)) i && "object" == typeof n && n && !H(n) &&
!G(n) && !(n instanceof RegExp) ? (e[s] || (e[s] = {}), U(e[s], n, r, i))
: e[s] = n
        }), e
    }

    function u(e, t) {
        return function() {
            return t.apply(e, arguments)
        }
    }

    function ca(e) {
        throw e
    }

    function da(e) {
```

Лістинг 1. Продовження

```
        if (!e) return e;
        var t = ba;
        return v(e.split("."), function(e) {
            t = t[e]
        }), t
    }

    function C(e, t, n, r) {
        return t = Error(t + "\nhttp://requirejs.org/docs/errors.html#" +
e), t.requireType = e, t.requireModules = r, n && (t.originalError = n),
t
    }

    function ga(e) {
        function n(e, t, n) {
            var r, i, s, o, u, a, f, l, t = t && t.split("/"),
                c = A.map,
                h = c && c["*"];
            if (e) {
                e = e.split("/"), i = e.length - 1, A.nodeIdCompat &&
Q.test(e[i]) && (e[i] = e[i].replace(Q, "")), "." === e[0].charAt(0) && t
&& (i = t.slice(0, t.length - 1), e = i.concat(e)), i = e;
                for (s = 0; s < i.length; s++)(o = i[s], "." === o) ?
(i.splice(s, 1), s -= 1) : ".." === o && 0 !== s && (1 != s || ".." !==
i[2]) && ".." !== i[s - 1] && 0 < s && (i.splice(s - 1, 2), s -= 2);
                e = e.join("/")
            }
            if (n && c && (t || h)) {
                i = e.split("/"), s = i.length;
                e: for (; 0 < s; s -= 1) {
                    u = i.slice(0, s).join("/");
                    if (t)
                        for (o = t.length; 0 < o; o -= 1)
                            if (n = m(c, t.slice(0, o).join("/")))
                                if (n = m(n, u)) {
                                    r = n, a = s;
                                    break e
                                }! f && h && m(h, u) && (f = m(h, u), l =
s)
                }!r && f && (r = f, a = l), r && (i.splice(0, a, r), e =
i.join("/"))
            }
            return (r = m(A.pkgs, e)) ? r : e
        }

        function r(e) {
            z && v(document.getElementsByTagName("script"), function(t) {
                if (t.getAttribute("data-requiremodule") === e &&
t.getAttribute("data-requirecontext") === x.contextName) return
t.parentNode.removeChild(t), !0
            })
        }

        function i(e) {
            var t = m(A.paths, e);
            if (t && H(t) && 1 < t.length) return t.shift(),
x.require.undef(e), x.makeRequire(null, {
                skipMap: !0
```

Лістинг 1. Продовження

```
    })([e]), !0
        }

        function s(e) {
            var t, n = e ? e.indexOf("!") : -1;
            return -1 < n && (t = e.substring(0, n), e = e.substring(n +
1, e.length)), [t, e]
        }

        function o(e, t, r, i) {
            var o, u, a = null,
                f = t ? t.name : null,
                l = e,
                c = !0,
                h = "";
            return e || (c = !1, e = "_@r" + (q += 1)), e = s(e), a =
e[0], e = e[1], a && (a = n(a, f, i), u = m(j, a)), e && (a ? h = u &&
u.normalize ? u.normalize(e, function(e) {
                return n(e, f, i)
            }) : -1 === e.indexOf("!") ? n(e, f, i) : e : (h = n(e, f,
i), e = s(h), a = e[0], h = e[1], r = !0, o = x.nameToUrl(h))), r = a &&
!u && !r ? "_unnormalized" + (W += 1) : "", {
                prefix: a,
                name: h,
                parentMap: t,
                unnormalized: !!r,
                url: o,
                originalName: l,
                isDefine: c,
                id: (a ? a + "!" + h : h) + r
            }
        }

        function a(e) {
            var t = e.id,
                n = m(O, t);
            return n || (n = O[t] = new x.Module(e)), n
        }

        function f(e, n, r) {
            var i = e.id,
                s = m(O, i);
            t(j, i) && (!s || s.defineEmitComplete) ? "defined" === n &&
r(j[i]) : (s = a(e), s.error && "error" === n) ? r(s.error) : s.on(n, r)
        }

        function l(e, t) {
            var n = e.requireModules,
                r = !1;
            t ? t(e) : (v(n, function(t) {
                if (t = m(O, t)) t.error = e, t.events.error && (r = !0,
t.emit("error", e))
            }), !r) && g.onError(e)
        }

        function c() {
            R.length && (ha.apply(P, [P.length, 0].concat(R)), R = [])
        }
```

Лістинг 1. Продовження

```
function h(e) {
            delete O[e], delete _[e]
        }

        function p(e, t, n) {
            var r = e.map.id;
            e.error ? e.emit("error", e.error) : (t[r] = !0, v(e.depMaps,
function(r, i) {
                var s = r.id,
                    o = m(O, s);
                o && !e.depMatched[i] && !n[s] && (m(t, s) ?
(e.defineDep(i, j[s]), e.check()) : p(o, t, n))
            }), n[r] = !0)
        }

        function d() {
            var e, t, n = (e = 1e3 * A.waitSeconds) && x.startTime + e <
(new Date).getTime(),
                s = [],
                o = [],
                u = !1,
                a = !0;
            if (!E) {
                E = !0, B(_, function(e) {
                    var f = e.map,
                        l = f.id;
                    if (e.enabled && (f.isDefine || o.push(e), !e.error))
                        if (!e.inited && n) i(l) ? u = t = !0 :
(s.push(l), r(l));
                        else if (!e.inited && e.fetched && f.isDefine &&
(u = !0, !f.prefix)) return a = !1
                });
                if (n && s.length) return e = C("timeout", "Load timeout
for modules: " + s, null, s), e.contextName = x.contextName, l(e);
                a && v(o, function(e) {
                    p(e, {}, {})
                }), (!n || t) && u && (z || ea) && !L && (L =
setTimeout(function() {
                    L = 0, d()
                }, 50)), E = !1
            }
        }

        function y(e) {
            t(j, e[0]) || a(o(e[0], null, !0)).init(e[1], e[2])
        }

        function b(e) {
            var e = e.currentTarget || e.srcElement,
                t = x.onScriptLoad;
            return e.detachEvent && !Y ?
e.detachEvent("onreadystatechange", t) : e.removeEventListener("load", t,
!1), t = x.onScriptError, (!e.detachEvent || Y) &&
e.removeEventListener("error", t, !1), {
                node: e,
                id: e && e.getAttribute("data-requiremodule")
```

Лістинг 2. eventBinders.js

```
define("newWidgets/widgets/launchStatisticsChart/LaunchStatisticsChartSetti
ngs", ["require", "jquery", "underscore", "localization", "util",
"defectType/SingletonDefectTypeCollection"], function(e) {
        "use strict";
        var t = e("jquery"),
            n = e("underscore"),
            r = e("localization"),
            i = e("util"),
            s = e("defectType/SingletonDefectTypeCollection"),
            o = function() {
                var e = [{
                        name: r.launchesHeaders.total,
                        value: "statistics$executions$total"
                    }, {
                        name: r.launchesHeaders.passed,
                        value: "statistics$executions$passed"
                    }, {
                        name: r.launchesHeaders.failed,
                        value: "statistics$executions$failed"
                    }, {
                        name: r.launchesHeaders.skipped,
                        value: "statistics$executions$skipped"
                    }],
                    t = {
                        product_bug: {
                            name: r.launchesHeaders.total + " " +
r.filterNamePluralById.product_bug,
                            value: "statistics$defects$product_bug$total"
                        },
                        automation_bug: {
```

Лістинг 2. Продовження

```
name: r.launchesHeaders.total + " " +
r.filterNamePluralById.automation_bug,
                            value:
"statistics$defects$automation_bug$total"
                        },
                        system_issue: {
                            name: r.launchesHeaders.total + " " +
r.filterNamePluralById.system_issue,
                            value: "statistics$defects$system_issue$total"
                        },
                        no_defect: {
                            name: r.launchesHeaders.total + " " +
r.filterNamePluralById.no_defect,
                            value: "statistics$defects$no_defect$total"
                        }
                    },
                    o = function() {
                        var e = new s,
                            r = "statistics$defects$",
                            o = {},
                            u = e.toJSON(),
                            a = i.getIssueTypes(),
                            f = [];
                        return n.each(a, function(e) {
                            o[e] = {}
                        }), n.each(u, function(e) {
                            var t = e.typeRef.toLowerCase(),
                                n = r + t + "$" + e.locator;
                            o[t][n] = e.longName
                        }), n.each(o, function(e, r) {
```

Лістинг 2. Продовження

```
                var i = Object.keys(e).length > 1;
                        i && f.push(t[r]), n.each(e, function(e, t) {
                            f.push({
                                name: e,
                                value: t,
                                isSubOption: i
                            })
                        })
                    }), f
                };
            return e.concat(o())
        };
    return {
        getConfig: function() {
            return {
                gadget_name: r.widgets.statisticsChart,
                img: "launch-statistics-line-chart.svg",
                description: r.widgets.statisticsChartDescription,
                widget_type: "trends_chart",
                hasPreview: !0
            }
        },
        getSettings: function() {
            var e = t.Deferred();
            return (new s).ready.done(function() {
                e.resolve([{
                    control: "filters",
                    options: {}
                }, {
```

Лістинг 2. Продовження

```
                    control: "dropDown",
                    options: {
                        label: r.widgets.widgetCriteria,
                        items: o(),
                        placeholder: r.wizard.criteriaSelectTitle,
                        multiple: !0,
                        getValue: function(e, t) {
                            var r = e.getContentFields();
                            return r[0] ? r :
n.map(t.model.get("items"), function(e) {
                                return e.value
                            })
                        },
                        setValue: function(e, t) {
                            t.setContentFields(e)
                        }
                    }
                }, {
                    control: "input",
                    options: {
                        name: r.widgets.items,
                        min: 1,
                        max: 150,
                        def: 50,
                        numOnly: !0,
                        action: "limit"
                    }
                }, {
                    control: "switcher",
```

Лістинг 2. Продовження

```
            options: {
        items: [{
            name: r.widgets.launchMode,
            value: "launch"
        }, {
            name: r.widgets.timelineMode,
            value: "timeline"
        }],
        action: "switch_timeline_mode"
    }
}, {
    control: "switcher",
    options: {
        items: [{
            name: r.widgets.areaChartMode,
            value: "areaChartMode"
        }, {
            name: r.widgets.barMode,
            value: "barMode"
        }],
        action: "switch_view_mode"
    }
}, {
    control: "checkbox",
    options: {
        label: r.widgets.zoomWidgetArea,
        getValue: function(e, t) {
            var n = e.getWidgetOptions();
            return n.zoom ? !0 : !1
```

Лістинг 2. Продовження

```
                            setValue: function(e, t) {
                                var n = t.getWidgetOptions();
                                e ? n.zoom = [] : delete n.zoom,
t.setWidgetOptions(n)
                            },
                            beta: !0
                        }
                    }, {
                        control: "static",
                        options: {
                            action: "metadata_fields",
                            fields: ["name", "number", "start_time"]
                        }
newWidgets/widgets/launchStatisticsChart/LaunchStatisticsChartSettings"],
function(e) {
        "use strict";
        var t =
e("newWidgets/widgets/launchStatisticsChart/LaunchStatisticsChartView"),
            n =
e("newWidgets/widgets/launchStatisticsChart/LaunchStatisticsChartSettings")
;
        return t.getConfig = n.getConfig, t.getSettings = n.getSettings, t
                                        }),
 define("newWidgets/widgets/overallStatistics/OverallStatisticsWidgetView",
"newWidgets/_C3ChartWidgetView",
"defectType/SingletonDefectTypeCollection",
"filters/SingletonLaunchFilterCollection", "localization", "d3", "c3",
"app"], function(e) {
        "use strict";
        var t = e("jquery"),
            n = e("underscore"),
```

Лістинг 2. Продовження

```
            r = e("util"),
            i = e("newWidgets/_C3ChartWidgetView"),
            s = e("defectType/SingletonDefectTypeCollection"),
            o = e("filters/SingletonLaunchFilterCollection"),
            u = e("localization"),
            a = e("d3"),
            f = e("c3"),
            l = e("app"),
            c = l.getInstance(),
            h = i.extend({
                templateDonut: "tpl-widget-overall-statistics-donut",
                templatePanel: "tpl-widget-overall-statistics-trend",
                templatePanelItem: "tpl-widget-overall-statistics-trend-
item",
                className: "overall-statistics",
                render: function() {
                    var e, i, u, a, f = {};
                    this.isPreview && this.$el.addClass("preview-view");
                    if (!this.isDataExists()) {
                        this.addNoAvailableBock();
"statistics$executions$total"]), n.each(this.model.getContentFields(),
function(e) {
 n.each(a, function(t, n) {
                            n === e && (f[n] = t)
                        })
                    });
                    if (i.statistics$executions$total === 0 || n.isEmpty(f)
&& n.isEmpty(u)) {
                        this.addNoAvailableBock();
                        return
```

Лістинг 2. Продовження

```
                position: function(e, t, n, r) {
                    var i = a.mouse(o.element)[0] - t / 2,
                        s = a.mouse(o.element)[1] - n;
                    return {
                        top: s - 8,
                        left: i
                    }
                },
                contents: function(e, t, n, r) {
                    var i, o, a =
e[0].id.split("$")[e[0].id.split("$").length - 1];
                    if (~["statistics$executions$passed",
"statistics$executions$failed", "legend").insert("div",
".legenter().append("span").attr("data-id", function(e) {
                        return e
                    }).html(function(e) {
                        var t, n, r = e.split("$")[e.split("$").length -
1];
                        if (~["statistics$executions$passed",
"statistics$executions$failed",
"statistics$executions$skipped"].indexOf(e)) t = u.launchesHeaders[r];
                        else {
                            n =
"legend").insert("div", ".legenter().append("span").attr("data-id",
function(e) {
                        return e
                    }).html(function(e) {
                        var t, n, r = e.split("$")[e.split("$").length -
1];
                        if (~["statistics$executions$passed",
"statistics$executions$failed",
```

Лістинг 2. Продовження

```
if (~["statistics$executions$passed", "statistics$executions$failed",
"statistics$executions$skipped"].indexOf(e)) t = u.launchesHeaders[r];

                        else {

                            n =
s.defetTypesCollection.getDefectByLocator(r);

                            if (!n) return '<div class="invalid-color-
mark"></div><span class="invalid">' + r + "</span>";

                            t = n.get("longName")

                        }

                        return '<div class="color-mark"></div>' + t

                    }).each(function(e) {

                        ~s.hiddenItems.indexOf(e) && t(".color-mark",
t(this)).addClass("unchecked"), a.select(this).select(".color-
mark").style("background-color", o.color(e))

                    }).on("mouseover", function(e) {

                        o.focus(e)

                    }).on("mouseout", function(e) {

                        o.revert()

                    }).on("click", function(e) {

                        c.trackingDispatcher.trackEventNumber(342),
t(".color-mark", t(this)).toggleClass("unchecked"), o.toggle(e)

                    }), this.hiddenItems && o.hide(this.hiddenItems),
a.select(o.element).select(".legend").append("div").attr("class", "legend-
gradient").append("div").attr("class", "legend-border"), m =
r.setupBaronScroll(t("[data-js-legend-wrapper]", e)),
```

Лістинг 3. Clastering.java

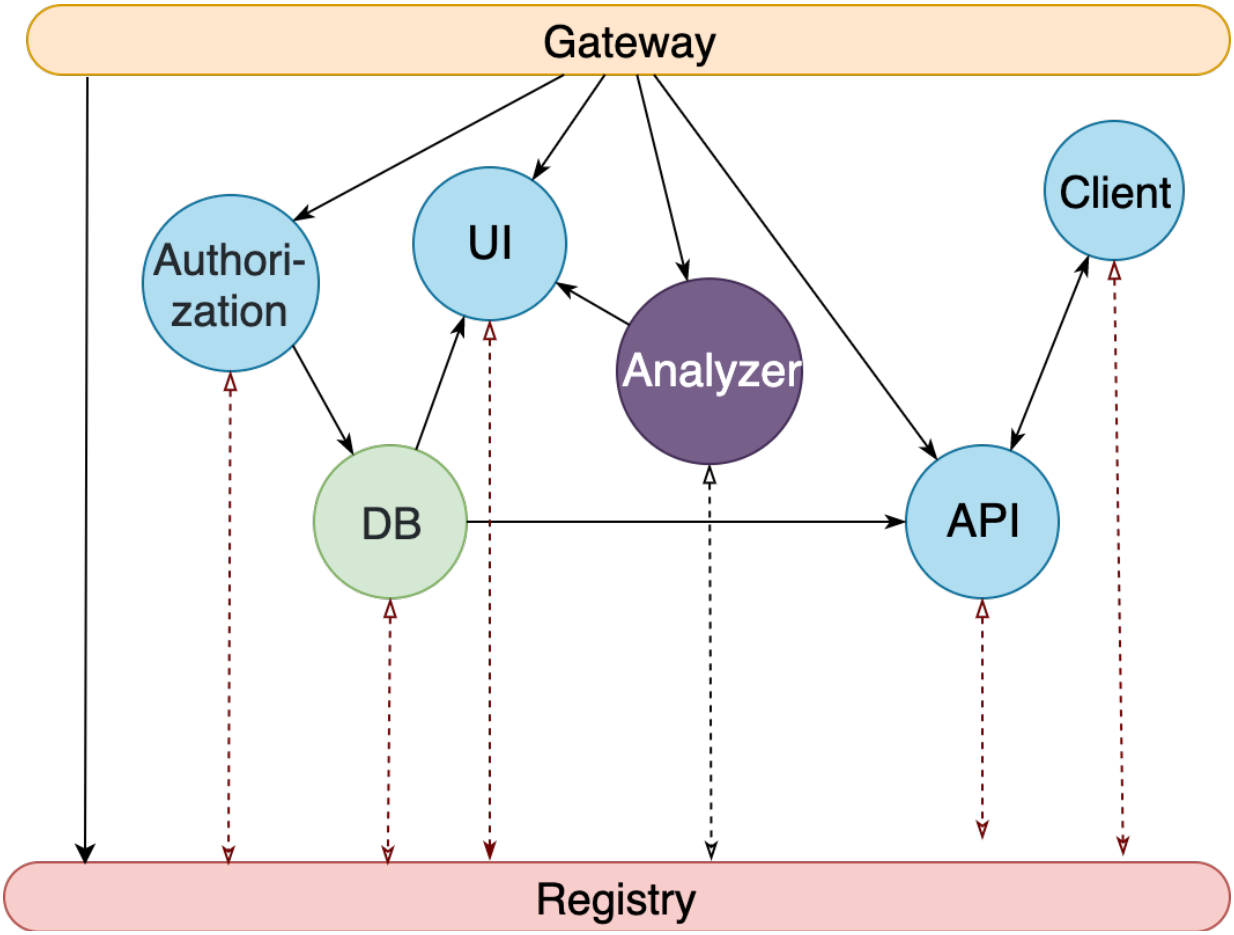
```
import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;


import weka.classifiers.Classifier;

import weka.classifiers.lazy.IBk;

import weka.core.Instance;

import weka.core.Instances;

private int determineK () {
size = data.size();
String sizeString = Integer.toString( size ) ;
sizeDouble = Double.parseDouble( sizeString );
root = Math.sqrt( sizeDouble );
double rawK = root / 2 ;
int num = Math.round( ( float )rawK ) ;
if ( num%2 != 0 ) {
return num ;
}
else {
return num - 1 ;
}
}

for (int index = 0; index < trainingData.size(); index++) {
    float distance = getDistanceBetween(dataPoint,
trainingData.get(index));
    distances.add(distance);
    distancesClone.add(distance);
}


Collections.sort(distances, new Comparator<Float>() {
    @Override
    public int compare(Float o1, Float o2) {
        return o1.compareTo(o2);
    }
});
```

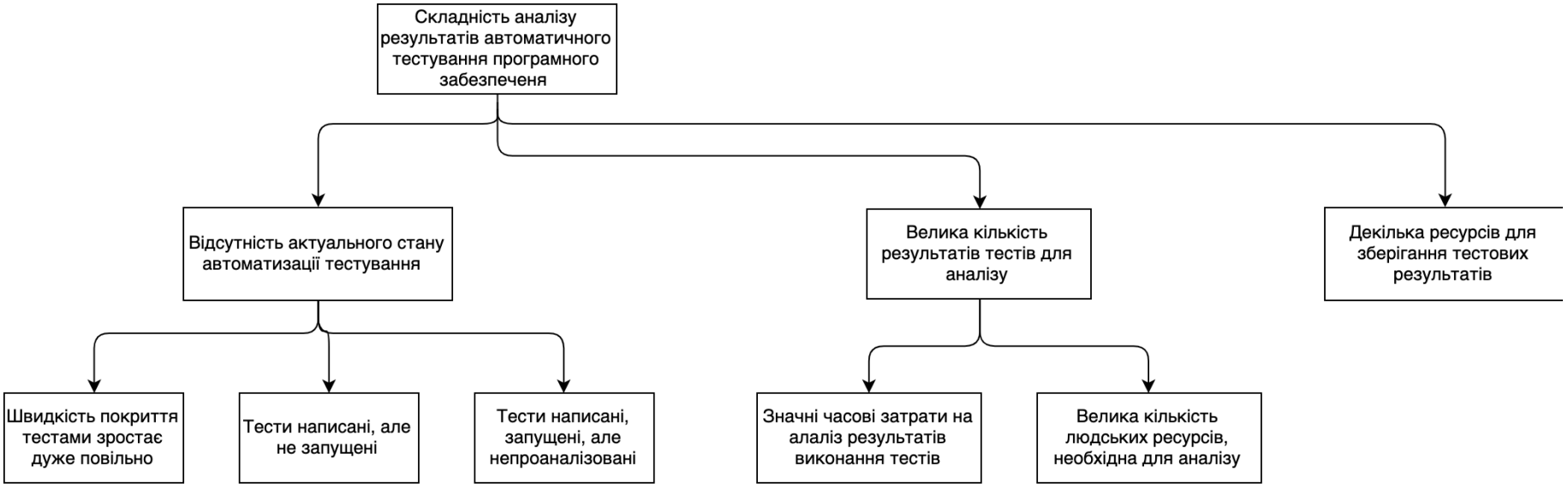
Лістинг 3. Продовження

```
int K = determineK() ;
List<Float >shortestDistances= distances.subList( 0 , K  ) ;
for ( float element : shortestDistances ) {
    Integer indexOnClone = distancesClone.indexOf(element);
    float[] nearestNeighbour = trainingData.get(indexOnClone);
    if (nearestNeighbour [ 3] == 1) {
        CLASS_1.add( nearestNeighbour ) ;
    }
    else if (nearestNeighbour [ 3] == 2) {
        CLASS_2.add( nearestNeighbour ) ;
    }
}
if ( CLASS_1.size() > CLASS_2 .size() ){
    return CLASS_1
}
else {
    return CLASS_2}
```

Додаток 2

Копії графічних матеріалів


Gateway
Authori-
zation
UI
Analyzer
Client
DB
API
Registry


Складність аналізу результатів автоматичного тестування програмного забезпеченя
Відсутність актуального стану автоматизації тестування
Велика кількість результатів тестів для аналізу
Декілька ресурсів для зберігання тестових результатів
Швидкість покриття тестами зростає дуже повільно
Тести написані, але не запущені
Тести написані, запущені, але непроаналізовані
Значні часові затрати на алаліз результатів виконання тестів
Велика кількість людських ресурсів, необхідна для аналізу

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

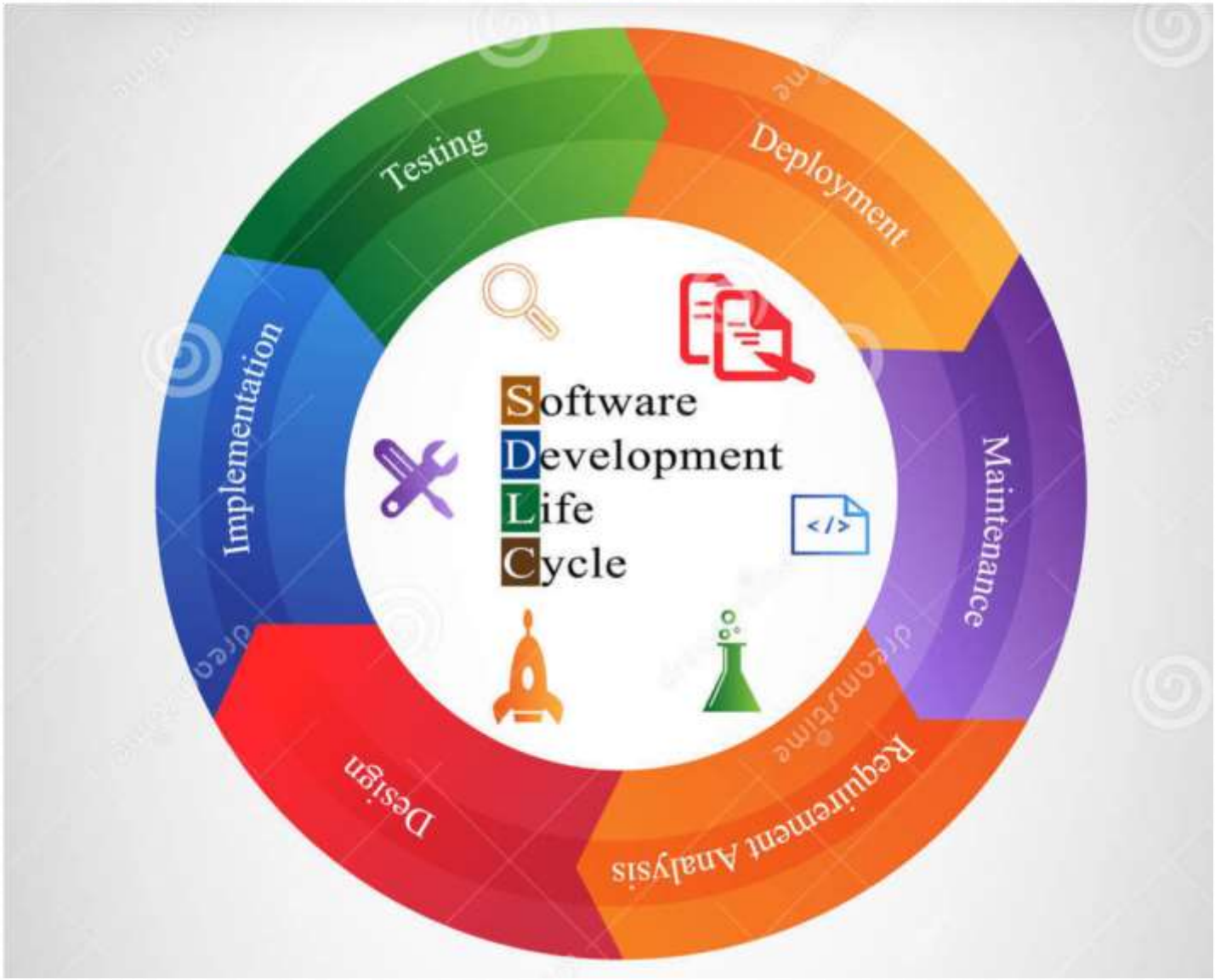
МЕТОД МАШИННОГО НАВЧАННЯ ДЛЯ АНАЛІЗУ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Виконав: Лисенко О. О., КП-81мп

Науковий керівник: к.т.н., доцент Олещенко Л. М.

Київ – 2019

АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ



НАУКОВЕ ЗАВДАННЯ

УДОСКОНАЛИТИ МЕТОДИ АНАЛІЗУ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

МЕТА ДОСЛІДЖЕННЯ

ЗМЕНШИТИ ЧАС, НЕОБХІДНИЙ ДЛЯ АНАЛІЗУ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

МЕТА ДОСЛІДЖЕННЯ

СТВОРИТИ ПРОГРАМНИЙ МЕТОД ДЛЯ АВТОМАТИЗОВАНОГО АНАЛІЗУ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ АЛГОРИТМУ KNN.

ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕНЬ

- **ОБ’ЄКТОМ** ДОСЛІДЖЕННЯ Є ПРОЦЕС АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗА ДОПОМОГОЮ МЕТОДІВ МАШИННОГО НАВЧАННЯ.
- **ПРЕДМЕТОМ** ДОСЛІДЖЕННЯ Є АЛГОРИТМИ КЛАСТЕРИЗАЦІЇ ДЛЯ АНАЛІЗУ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

ТЕРМІНОЛОГІЯ

- ***STACK TRACE*** – СПИСОК МЕТОДІВ, ЯКІ БУЛИ ВИКЛИКАНІ ДО МОМЕНТУ, КОЛИ В ДОДАТКУ ВІДБУЛАСЯ ПОМИЛКА.
- ***КЛАСТЕРИЗАЦІЯ*** – ЗАДАЧА РОЗБИТТЯ ЗАДАНОЇ ВИБІРКИ ОБ'ЄКТІВ (СИТУАЦІЙ) НА ПІДМНОЖИНИ, ЯКІ НАЗИВАЮТЬСЯ КЛАСТЕРАМИ, ТАК, ЩОБ КОЖЕН КЛАСТЕР СКЛАДАВСЯ ЗІ СХОЖИХ ОБ'ЄКТІВ, А ОБ'ЄКТИ РІЗНИХ КЛАСТЕРІВ ІСТОТНО ВІДРІЗНЯЛИСЯ.
- ***TF*** (TERM FREQUENCY – ЧАСТОТА СЛОВА) – ВІДНОШЕННЯ ЧИСЛА ВХОДЖЕНЬ ОБРАНОГО СЛОВА ДО ЗАГАЛЬНОЇ КІЛЬКОСТІ СЛІВ ДОКУМЕНТА. ОЦІНЮЄТЬСЯ ВАЖЛИВІСТЬ СЛОВА У МЕЖАХ ОБРАНОГО ДОКУМЕНТА.
- ***IDF*** (INVERSE DOCUMENT FREQUENCY – ОБЕРНЕНА ЧАСТОТА ДОКУМЕНТА) – ІНВЕРСІЯ ЧАСТОТИ, З ЯКОЮ СЛОВО ЗУСТРІЧАЄТЬСЯ В ДОКУМЕНТАХ КОЛЕКЦІЇ.

ПРОБЛЕМАТИКА

- ЗНАЧНІ ЧАСОВІ ЗАТРАТИ НА АНАЛІЗ РЕЗУЛЬТАТІВ ВИКОНАННЯ ТЕСТІВ
- ВЕЛИКА КІЛЬКІСТЬ ЛЮДСКИХ РЕСУРСІВ, НЕОБХІДНИХ ДЛЯ АНАЛІЗУ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ
- СКЛАДНІСТЬ ОЦІНЮВАННЯ ПОКРИТТЯ ТЕСТАМИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПРОБЛЕМАТИКА

Покриття зростає повільніше, ніж планувалося	Реалізовані, але не запущені	Запущені, але не проаналізовані	Нестабільні, незв'язані з самим продуктом
	Нульова цінність	Нульова цінність	Нульова цінність

ЗАДАЧІ

1.АНАЛІЗ МЕТОДІВ КЛАСТЕРИЗАЦІЇ ТЕКСТУ РЕЗУЛЬТАТІВ ВИКОНАНОГО ТЕСТУ ЗА КАТЕГОРІЯМИ ПОМИЛОК ТА ЇХ ПОРІВНЯННЯ

2.ПРОПОЗИЦІЯ МЕТОДУ ДЛЯ АВТОМАТИЗОВАНОГО АНАЛІЗУ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.РОЗРОБЛЕННЯ ВЕБ-ДОДАТКУ З ВИКОРИСТАННЯМ ОБРАНОГО АЛГОРИТМУ

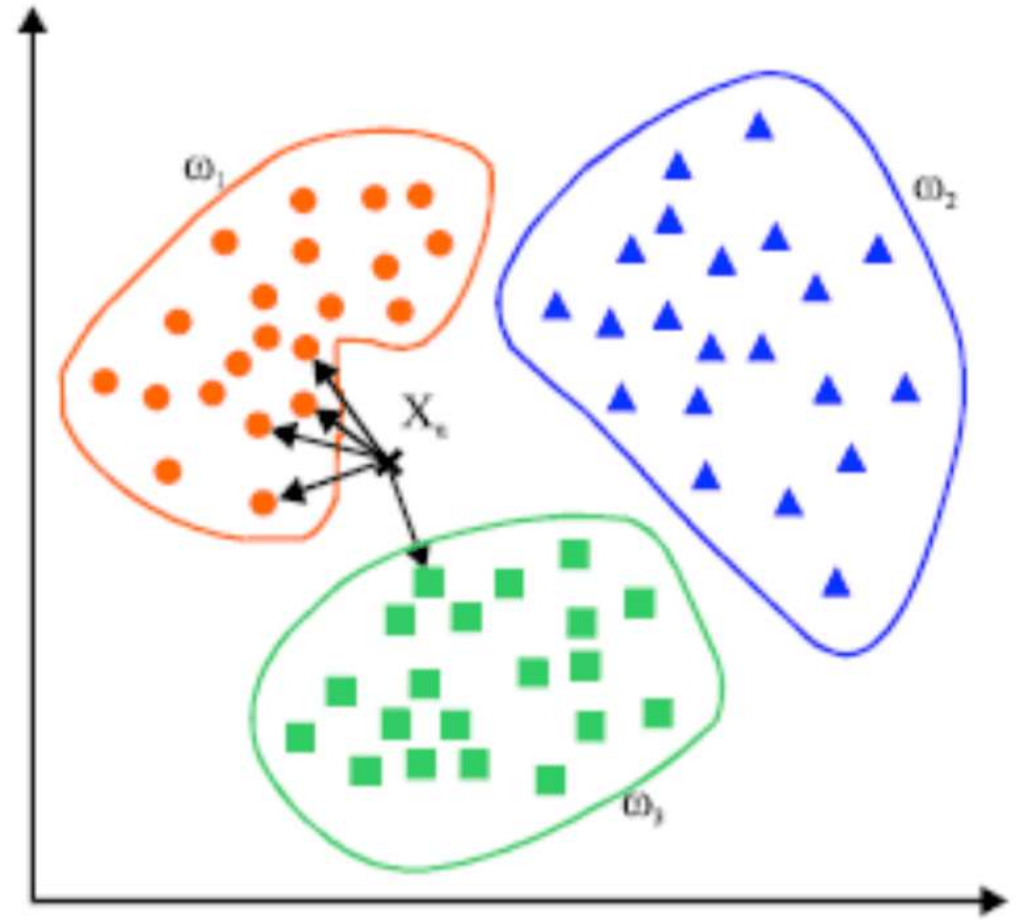
4.ТЕСТУВАННЯ ТА ПОРІВНЯННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

5.СТВОРЕННЯ БІЗНЕС-МОДЕЛІ ПРОГРАМНОГО ПРОДУКТУ.

МЕТОДИ КЛАСТЕРИЗАЦІЇ

Назва методу	Простота реалізації	Точність кластеризації	Швидкість виконання
Метод Байєса	-	+/-	+
Метод опорних векторів	-	+	+
Метод k-найближчих сусідів	+	+	-

МЕТОД *K*-НАЙБЛИЖЧИХ СУСІДІВ



РОБОТА АЛГОРИТМУ:

```
2019-09-02 07:55:37

java.lang.AssertionError: Invalid Upc Service Navigation
link redirection. expected [true] but found [false]
org.testng.Assert.fail(Assert.java:94)
org.testng.Assert.failNotEquals(Assert.java:513)
org.testng.Assert.assertTrue(Assert.java:42)
my.project.tests.checkLinksAreClickable(MainTest.java:61)
```

РОБОТА АЛГОРИТМУ:

```
java lang assertionError  invalid upc service navigation
link redirection  expected  true       found  false
org testng assert fail assert java
org testng assert failnotequals assert java
org testng assert asserttrue assert java
my project tests checkLinksareclickable maintest java
```

РОБОТА АЛГОРИТМУ:

```
        4          5          2          7          5
                                        expected   found
                                        expected   found
                              Service   expected
AssertionError   Invalid
                 Invalid                expected
                                        expected   found
AssertionError   Invalid                           found
                 Invalid
AssertionError                          expected
AssertionError   Invalid    Service     expected   found
```

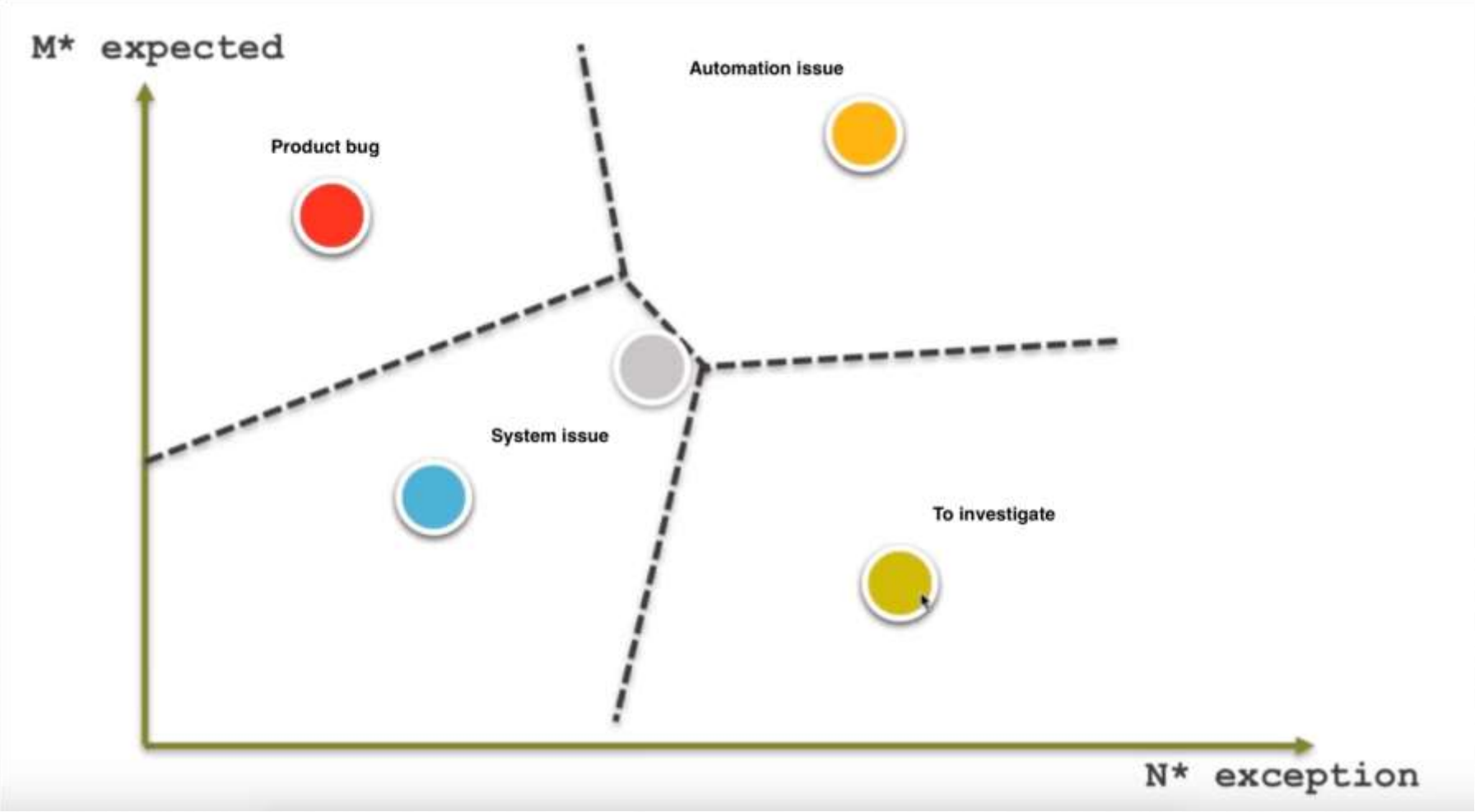
TF-IDF МЕТРИКА

$$IDF = log \frac{|D|}{|d_i \supset t_i|}$$

$|D|$ — кількість документів колекції;

$|(d_i \supset t_i)|$ — кількість документів, у яких зустрічається слово t_i (коли $n_i \neq 0$).

РОБОТА АЛГОРИТМУ:



НЕДОЛІКИ ІСНУЮЧИХ АНАЛОГІВ

- ВІДСУТНІСТЬ МОЖЛИВОСТІ АВТОМАТИЧНОЇ КАТЕГОРИЗАЦІЇ РЕЗУЛЬТАТІВ АВТОТЕСТІВ
- ОБМЕЖЕНА КІЛЬКІСТЬ КАТЕГОРІЙ ДЛЯ КЛАСТЕРИЗАЦІЇ РЕЗУЛЬТАТІВ АВТОТЕСТІВ.
- ВІДСУТНІСТЬ ІСТОРІЇ ВИКОНАННЯ ТЕСТІВ

ТЕСТУВАННЯ МЕТОДУ

- UNIT ТЕСТИ
- РУЧНЕ ТЕСТУВАННЯ

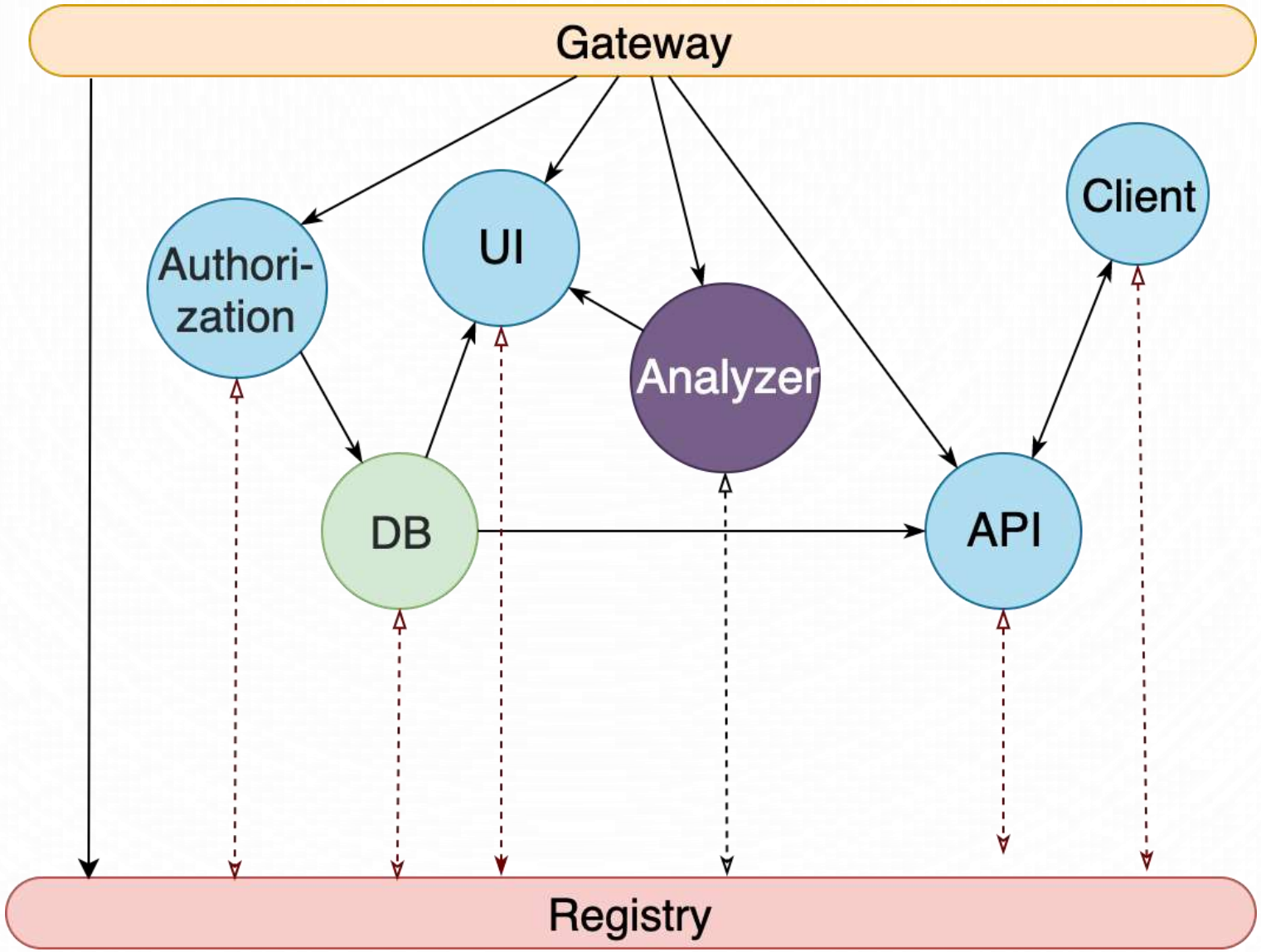
ПОРІВНЯННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Кількість провалених тестів	Метод k-найближчих сусідів	Метод опорних векторів	Метод Байєса
100	0.7483	0.7948	0.8126
200	0.8347	0.8589	0.7650
500	0.9078	0.8451	0.7607
1000	0.9820	0.9241	0.7520

РЕЗУЛЬТАТИ ТЕСТУВАННЯ МЕТОДУ

АНАЛІЗ 150 «ПРОВАЛЕНИХ» ТЕСТІВ ЗАЙНЯВ ПРИБЛИЗНО 4 ГОДИНИ, АЛГОРИТМ ВПОРАВСЯ З ЦІЄЮ ЗАДАЧЕЮ ЗА 20 ХВИЛИН, ЩО У 12 РАЗІВ ШВИДШЕ, НІЖ РУЧНИЙ АНАЛІЗ.

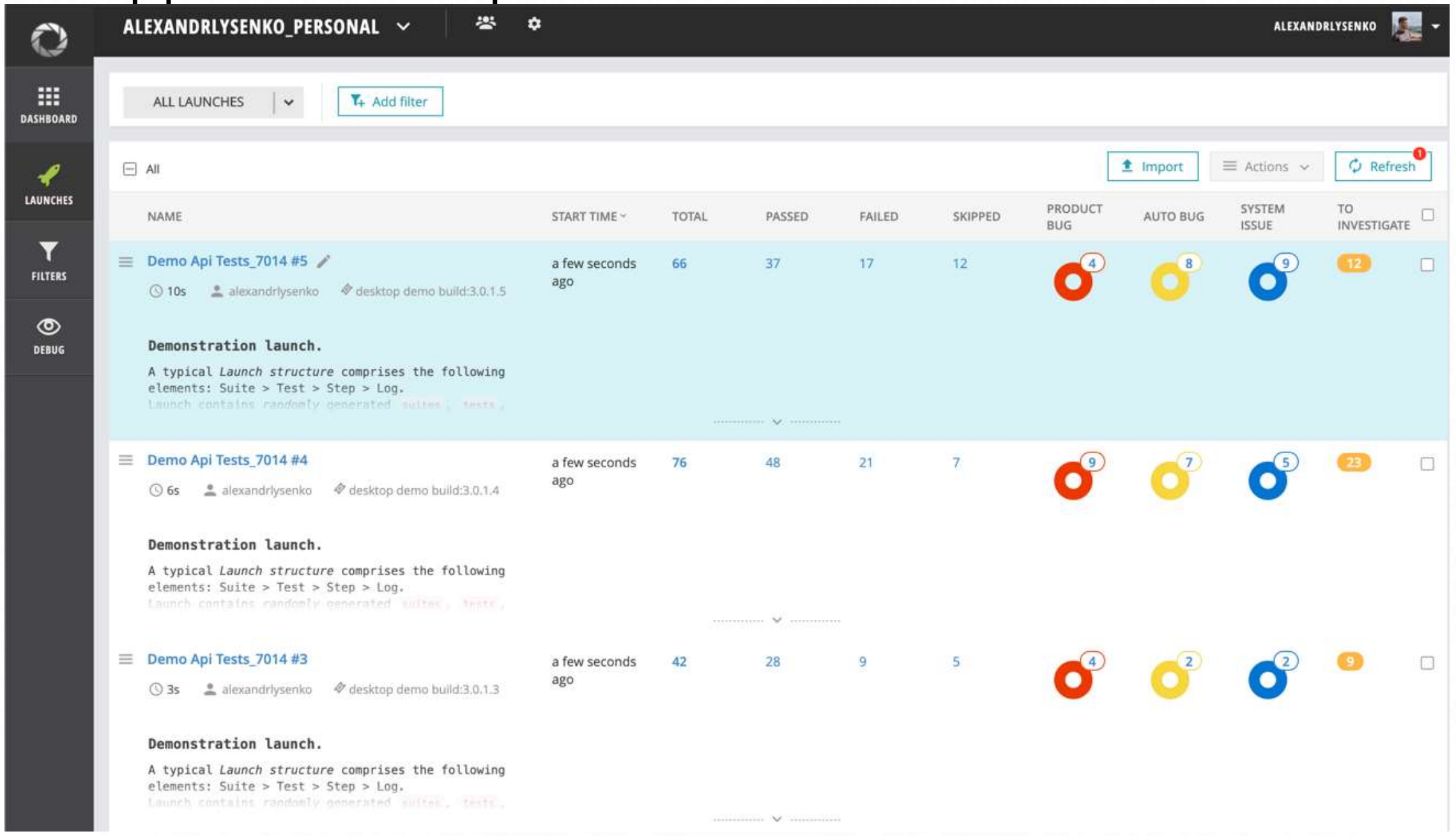
АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



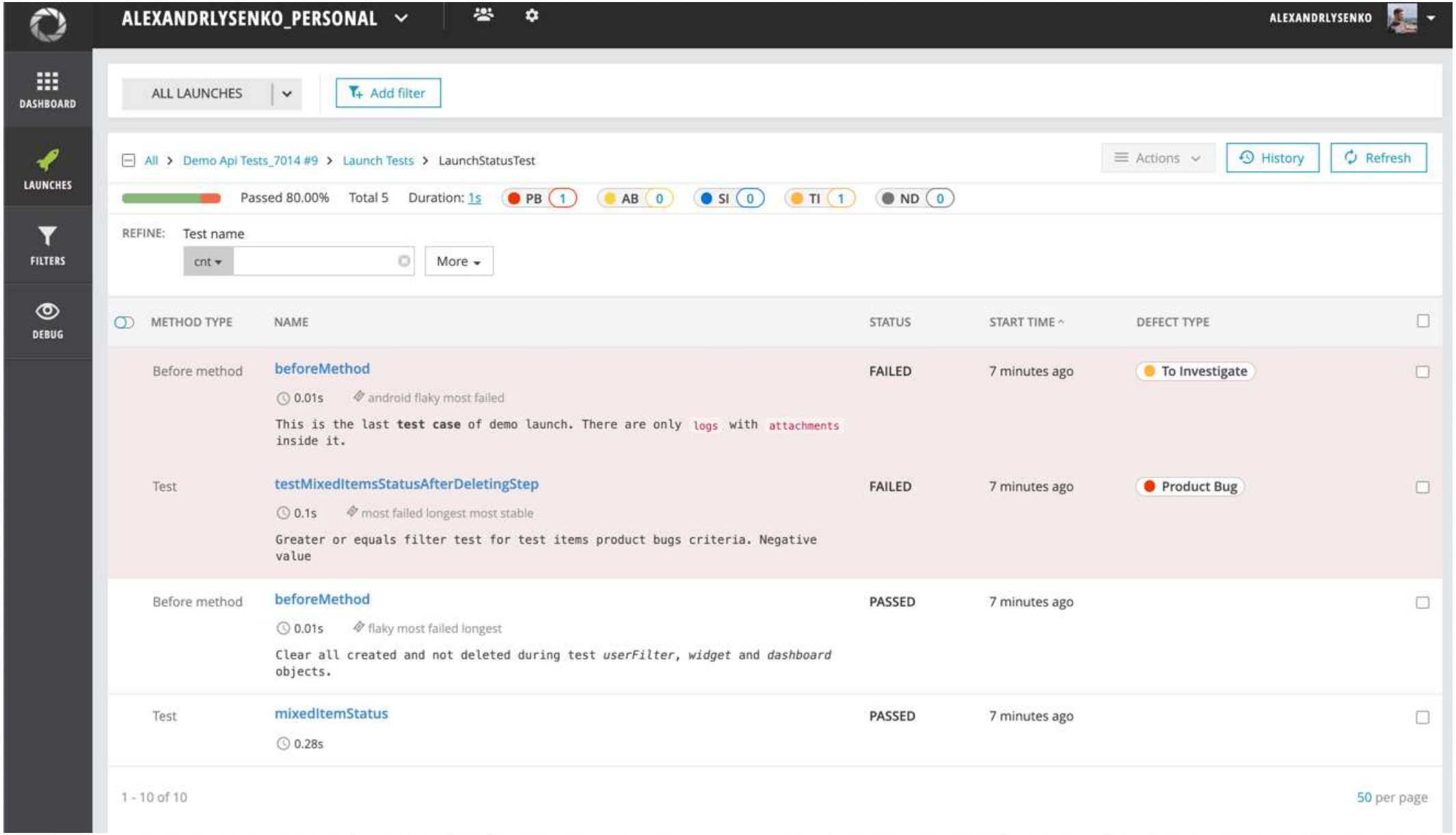
ВИКОРИСТАНІ ТЕХНОЛОГІЇ

- Java
- MongoDB
- Elasticsearch
- Angular
- HTML & CSS
- JavaScript
- Node.Js
- Jenkins
- Docker
- Git

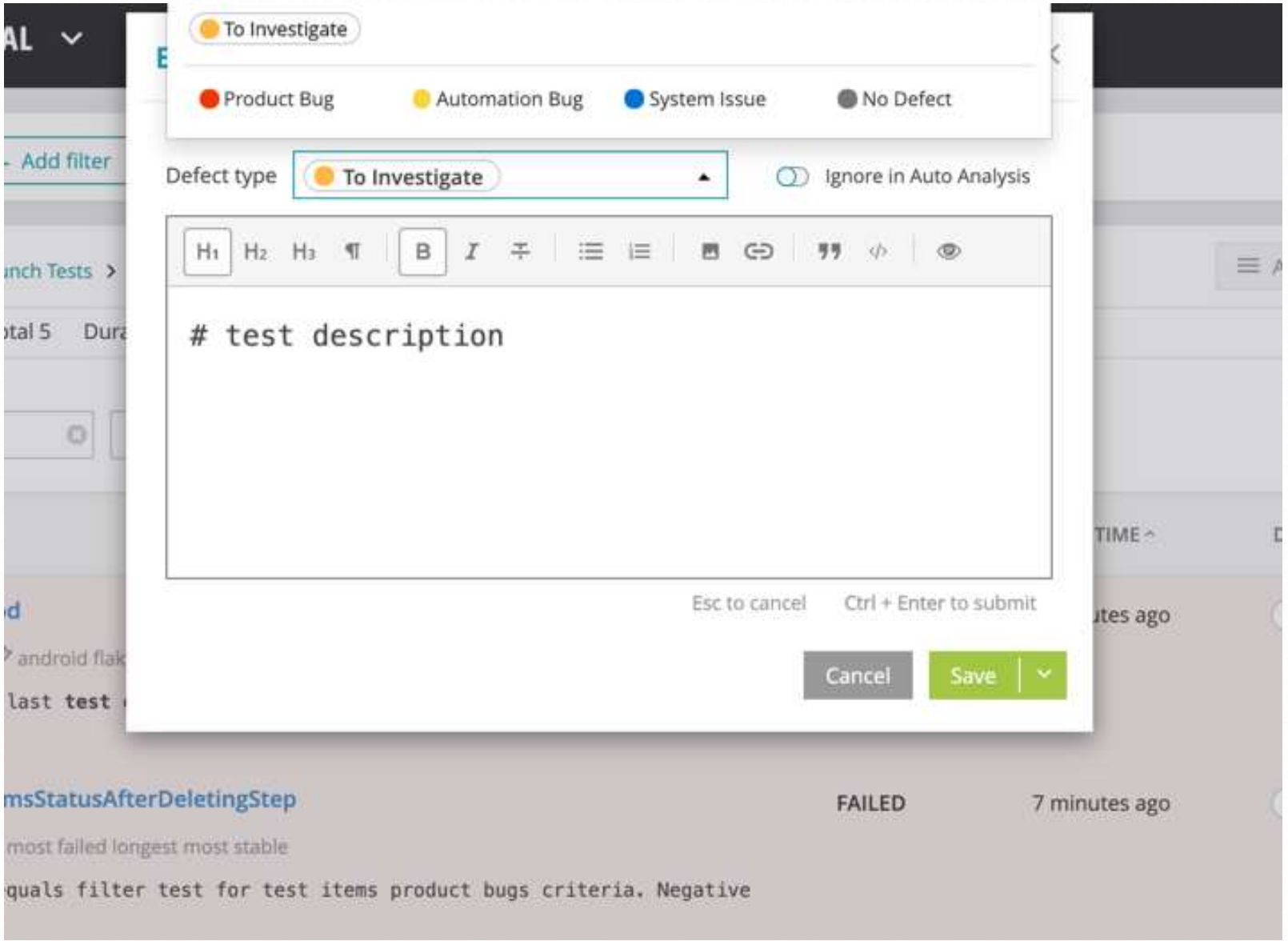
ДЕМОНСТРАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



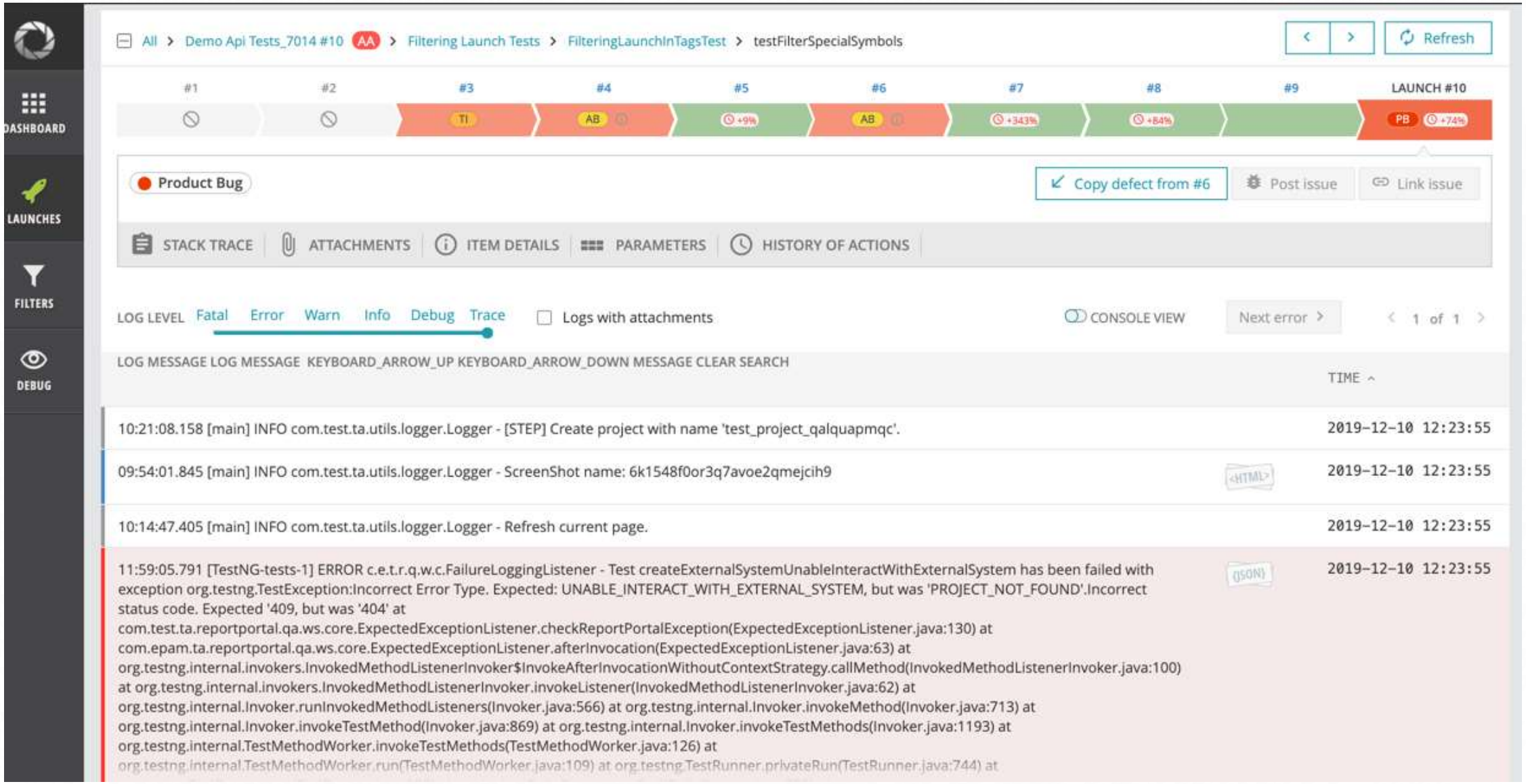
ДЕМОНСТРАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



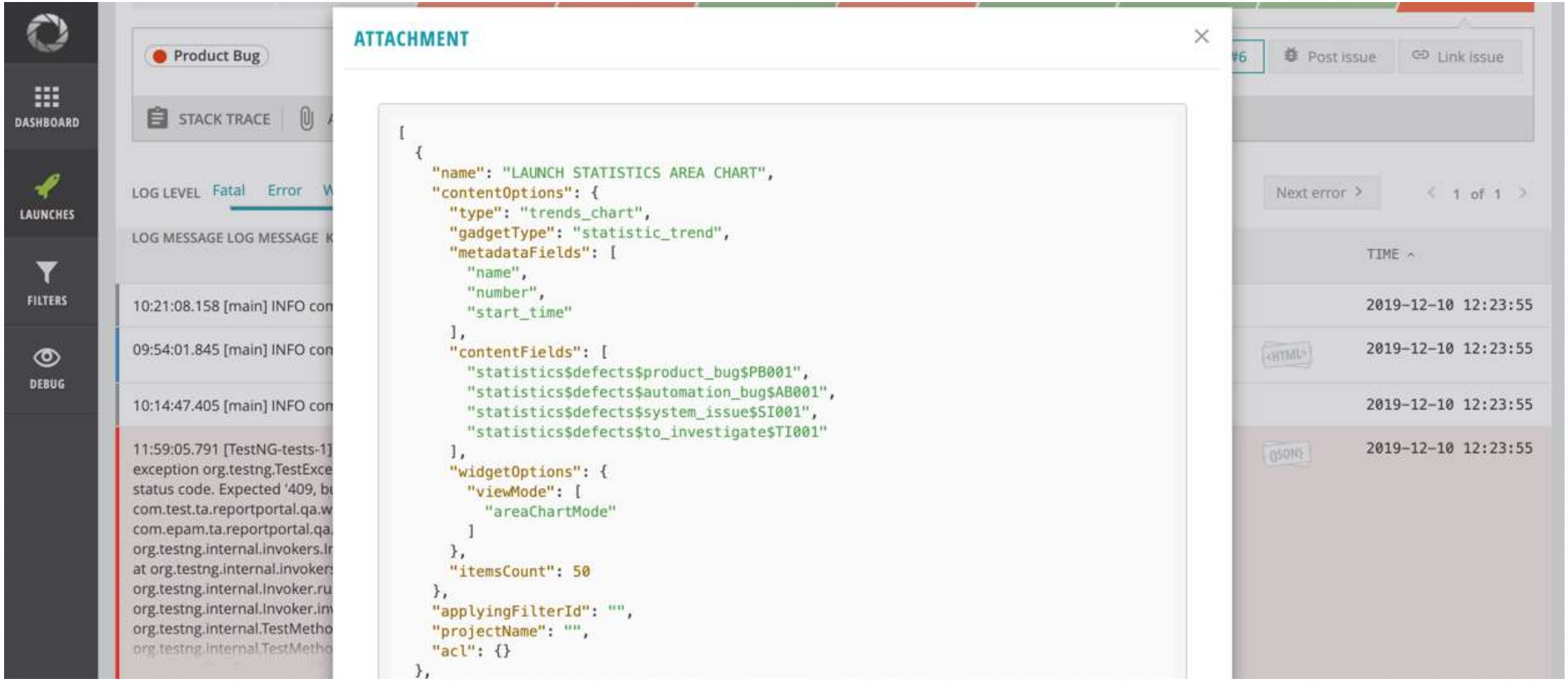
ДЕМОНСТРАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



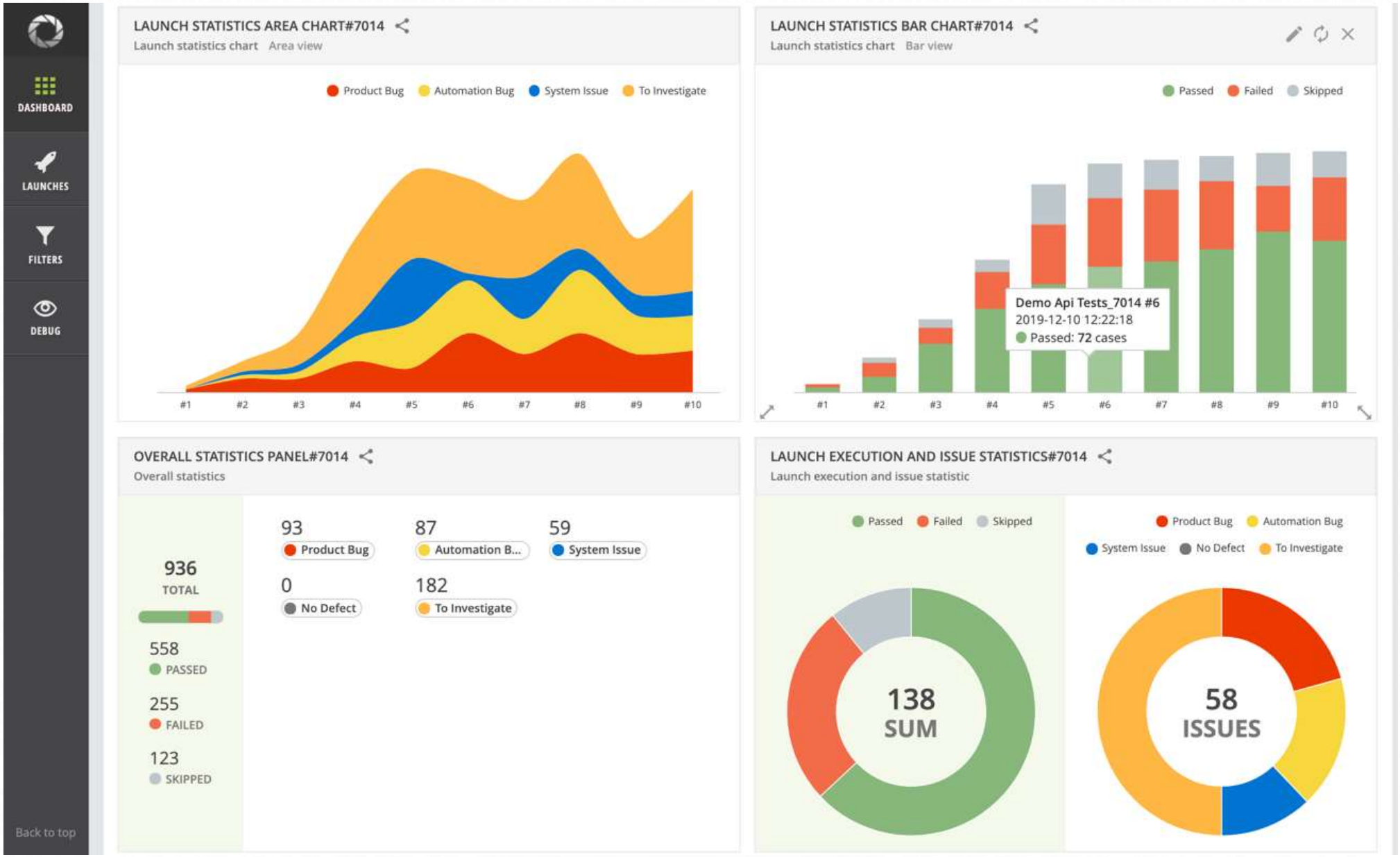
ДЕМОНСТРАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



ДЕМОНСТРАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



ДЕМОНСТРАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



НАУКОВА НОВИЗНА

ВПЕРШЕ ЗАПРОПОНОВАНО МЕТОД, ЯКИЙ, НА ВІДМІНУ ВІД ІСНУЮЧИХ РІШЕНЬ, ДОЗВОЛЯЄ АВТОМАТИЧНО АНАЛІЗУВАТИ РЕЗУЛЬТАТИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ЩО ЗНАЧНО ЗМЕНШУЄ ЧАС, НЕОБХІДНИЙ ДЛЯ АНАЛІЗУ ПОМИЛОК ТА ДОЗВОЛЯЄ ПЕРЕГЛЯДАТИ СТАН ВИКОНАННЯ ТЕСТІВ У РЕАЛЬНОМУ ЧАСІ.

БІЗНЕС-МОДЕЛЬ

Проблема	Рішення	Унікальна ціннісна пропозиція	Споживачі	Ключові метрики
• Значні часові затрати на аналіз результатів виконання тестів • Велика кількість людских ресурсів, необхідна для аналізу • Складність оцінювання покриття тестами у проекті	Програмне забезпечення, що дозволяє в реальному часі категоризувати результати тестування програмного забезпечення за допомогою машинного навчання	• Зменшення часу, потрібного на аналіз результатів тестування ПЗ • Зменшення людських ресурсів, необхідних для аналізу • Надання повної картини автоматизації тестування проекту	• Великі та середні компанії • Розробники • Компанії, що займаються виключно тестуванням	• Кількість проданих ліцензій • Кількість зроблених мікротранзакцій.

БІЗНЕС-МОДЕЛЬ

Канали	Структури витрат	Потоки доходів	Ключові метрики
• Соціальні мережі • Контекстна реклама	• Початковий персонал для розроблення програмного продукту, • Утримання персоналу для надання технічної підтримки • Заробітня платня та соціальні виплати • Оплата за оренду офісу	• Доходи від продажу ліцензій • Доходи з мікротранзакцій .	• Кількість проданих ліцензій • Кількість зроблених мікротранзакцій

ФІНАНСОВИЙ ПЛАН СТАРТАПУ

Доходи(тис.$)	Продаж ліцензій	Зроблені мікро транзакції	Довготривалі підписки	Всього
Сумарно за рік:	250	300	180	730

Витрати(тис.$)	Технічна підтримка	Оплата праці	Оренда та комунальні витрати	Реклама	Всього
Сумарно за рік:	10	220	25	12	267

АПРОБАЦІЯ

XII НАУКОВА КОНФЕРЕНЦІЯ МАГІСТРАНТІВ ТА АСПІРАНТІВ «ПРИКЛАДНА МАТЕМАТИКА ТА КОМП'ЮТИНГ» (ПМК-2019).

ВИСНОВОК

1. ПРОАНАЛІЗОВАНО МЕТОДИ КЛАСТЕРИЗАЦІЇ ДЛЯ АНАЛІЗУ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ ТА ВИДІЛЕНО ЇХ НЕДОЛІКИ.
2. ЗАПРОПОНОВАНО МЕТОД З ВИКОРИСТАННЯМ KNN-АЛГОРИТМУ, ЯКИЙ ЗА ДОПОМОГОЮ СТЕКТРЕЙСУ КЛАСТЕРИЗУЄ РЕЗУЛЬТАТИ АВТОТЕСТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.
3. СТВОРЕНО ВЕБ-ДОДАТОК, ЩО ДОЗВОЛЯЄ ПЕРЕГЛЯДАТИ СТАН ТЕСТУВАННЯ У ПРОЕКТІ ТА АНАЛІЗУВАТИ РЕЗУЛЬТАТИ ВИКОНАННЯ АВТОТЕСТІВ У РЕАЛЬНОМУ ЧАСІ.
4. ВИКОНАНО ТЕСТУВАННЯ ТА ПОРІВНЯННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ. АНАЛІЗ 150 «ПРОВАЛЕНИХ» ТЕСТІВ ВРУЧНУ ЗАЙНЯВ ПРИБЛИЗНО 4 ГОДИНИ, АЛГОРИТМ KNN ВИКОНАВ АНАЛІЗ ЗА 20 ХВИЛИН, ЩО У 12 РАЗІВ ШВИДШЕ, НІЖ РУЧНИЙ АНАЛІЗ, ТОБТО ЗАПРОПОНОВАНИЙ ПІДХІД Є ЕФЕКТИВНИМ.
5. РОЗРОБЛЕНО БІЗНЕС-МОДЕЛЬ ПРОГРАМНОГО ПРОДУКТУ.

ДЯКУЮ ЗА УВАГУ!